



DeepL

Subscribe to DeepL Pro to translate larger documents.  
Visit [www.DeepL.com/pro](https://www.DeepL.com/pro) for more information.

# 디지털 디자인 및 컴퓨터 아키텍처

사라 해리스 & 데이비드 해리스

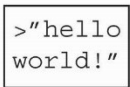


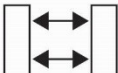
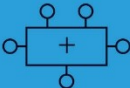
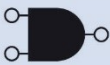
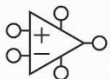

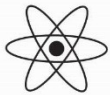
## 3장: 순차적 로

# 직 설계

수정 됨 유영환, 2023

# 3장 :: 주제

- 상태 요소
  - 바이스테이블 회로
  - SR 래치
  - D 래치
  - D 플립플롭
  - 변형
- 동기식 순차 로직
- 유한 상태 머신
  - Moore
  - Mealy
- 전이성
- 병렬 처리

Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

# 3장: 순차 논리

## 상태 요소

# 소개

- 순차 로직의 출력은 현재 *뿔* 이전 입력 값에 따라 달라지며 **메모리**가 있습니다.
- 몇 가지 정의:
  - **상태**: 향후 동작을 설명하는 데 필요한 회로에 대한 모든 정보
  - **래치와 플립플롭**: 한 비트의 상태를 저장하는 상

## 태 요소

- 동기식 순차 회로: 공통 클럭을 공유하는 플립  
플롭을 사용하는 순차 회로

# 순차 회로

- 이벤트에 순서 부여
- 기억력 보유(단기)
- 출력에서 입력으로 피드백을 사용하여 정보 저장

# 상태 요소

- **상태:** 향후 동작을 예측하는 데 필요한 회로의 이전 입력에 대한 모든 것
  - 일반적으로 캡처한 마지막 값인 1비트에 불과합니다.
- 상태 요소는 상태를 저장합니다.
  - 바이스테이블 회로



- SR 래치
- D 래치
- D 플립플롭

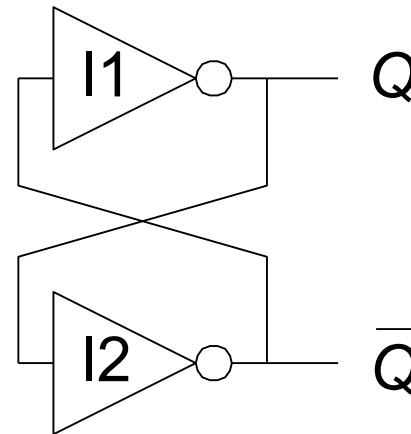
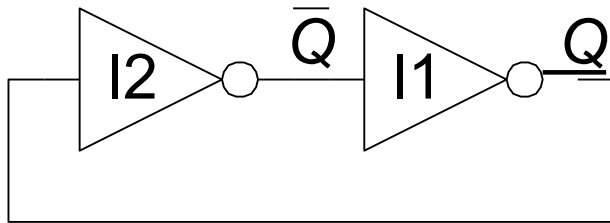
## 3장: 순차 논리

# 바이스테인블 회로

# 바이스테이블 회로

- 다른 상태 요소의 기본 구성 요소
- 두 가지 출력:  $Q$ ,  $\bar{Q}$
- 입력 없음

같은 회로!



## 백투백 인버터

## 크로스 커플링 인버터

# 바이스테이블 회로 분석

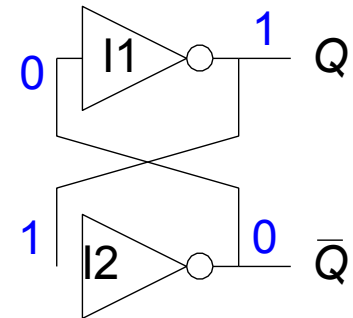
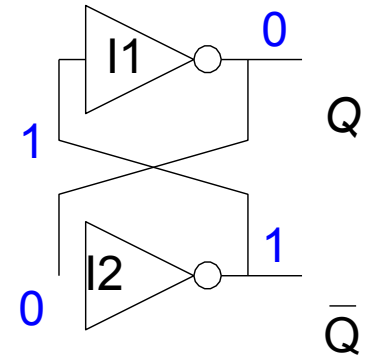
- 두 가지 경우를 생각해 보세요:

-  $Q = 0$ :

이면  $\bar{Q} = 1, Q = 0$ (일관성)

-  $Q = 1$ :

이면  $\bar{Q} = 0, Q = 1$ (일관성)



- 상태 변수 Q(또는  $\bar{Q}$ )에 상태 1비트를 저장합니다.

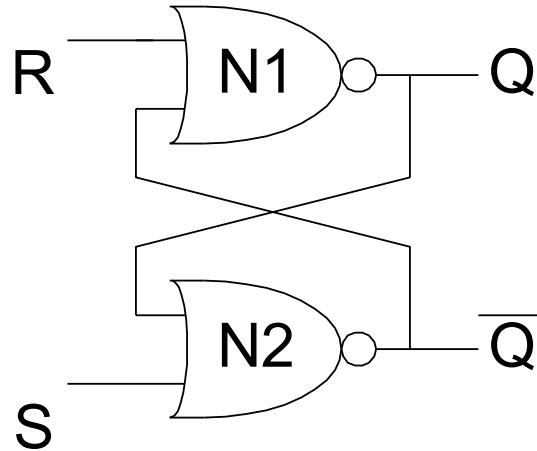
- 하지만 **상태를 제어할 입력이 없습니다.**

## 3장: 순차 논리

# SR 래치

# SR(설정/리셋) 래치

- SR 래치



- 네 가지 경우를 생각해 보세요:

- $S = 1, R = 0$

- $S = 0, R = 1$

- $S = 0, R = 0$



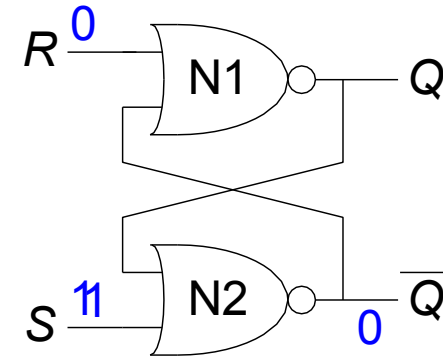
-  $S = 1, R = 1$

# SR 래치 분석

-  $S = 1, R = 0$ :

이면  $Q = 1, \overline{Q} = 0$

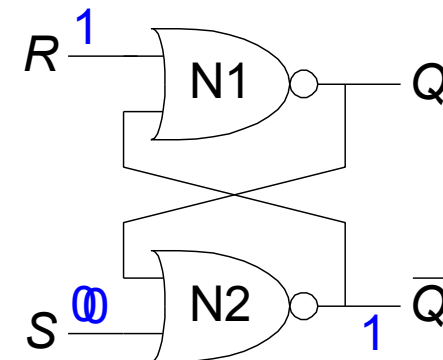
출력 설정



-  $S = 0, R = 1$ :

이면  $Q = 0, \overline{Q} = 1$

출력 재설정

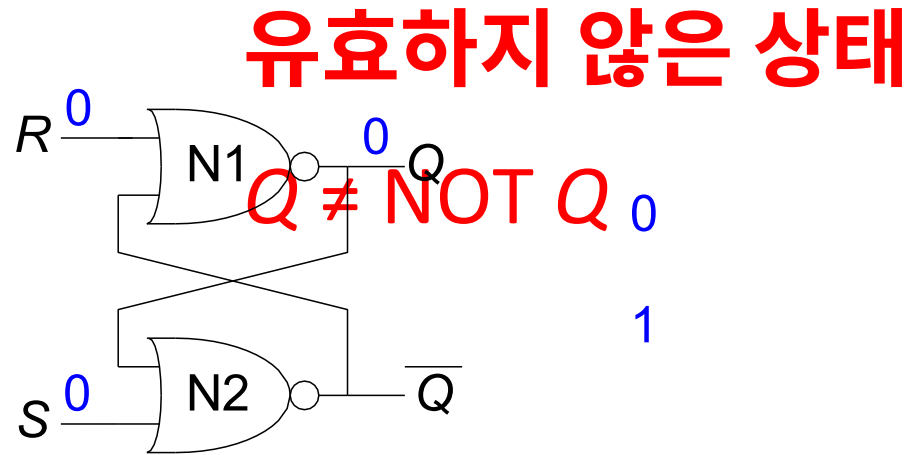


# SR 래치 분석

-  $S=0, R=0$ :

$$\underline{Q} = Q_{prev}$$

**메모리!**



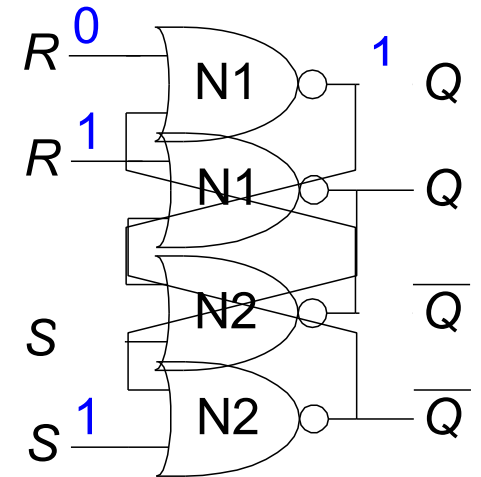
-  $S=1, R=1$ :

그러면  $Q = \overline{0}, Q$   
 $= 0$

0  
0  
0  
0

$Q_{prev} = 0$   
1

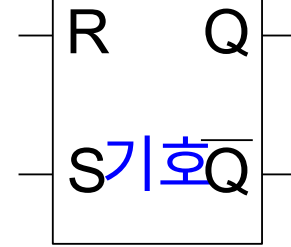
$Q_{prev} =$



# SR 래치

- **SR**은 래치 설정/재설정을 나타냅니다.
  - 한 비트의 상태( $Q$ )를 저장합니다.
- $S, R$  입력으로 어떤 값을 저장할지 제어합니다.
  - **설정**: 출력 1로 설정  
 $S = 1, R = 0, Q = 1$   
 $S = 0, R = 0, Q = Q_{prev}$
  - **초기화**: 출력을 0으로 설정  
 $S = 0, R = 1, Q = 0$
  - **메모리**: 가치 유지

## SR 래치



- 유효하지 않은 상태를 피하기 위해 무언가를 해야 합니다(  
 $S = R = 1$ )

## 3장: 순차 논리

# D 래치

# D 래치

- 두 개의 입력:  $CLK$ ,  $D$ 
  - $CLK$ : 출력 변경 시/기/제어
  - $D$ (데이터 입력): 출력이 무엇으로 변경되는지 제어합니다.
- 기능
  - $CLK = 1$ 인 경우,  
 $D$ 는  $Q$ (투명)로 전달됩니다.
  - $CLK = 0$   
인 경우,



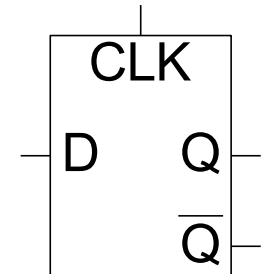
Q는 이전 값을 유지합니다(*불투명*).

- 다음과 같은 유효하지 않은 경우  
를 방지합니다.

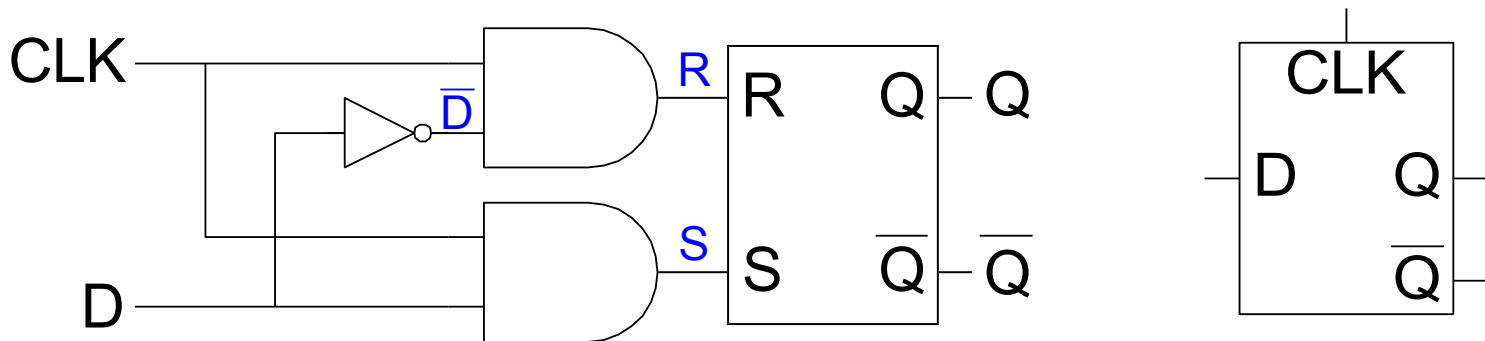
$$Q \neq \text{NOT } \bar{Q}$$

D 래치

기호



# D 래치 내부 회로



$CLK$	$D$	$\overline{D}$	$S$	$R$	$Q$	$\overline{Q}$
0	X	$\overline{X}$	0	0	$Q_{prev}$	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

# 3장: 순차 논리

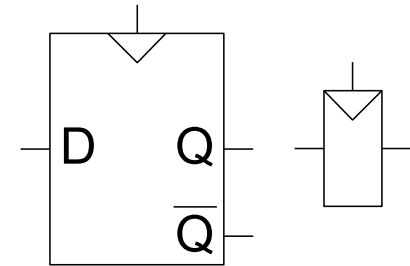
## D 플립플롭

# D 플립플롭

- 입력:  $CLK$ ,  $D$
- 기능:
  - $CLK$ 의 상승 가장자리에 있는 샘플  $D$ 
    - $CLK$ 가 0에서 1로 상승하는 경우,  $D$ 는  $Q$ 로 전달됩니다.
    - 그렇지 않으면  $Q$ 는 이전 값을 유지합니다.
  - $Q$ 는  $CLK$ 의 상승 에지에서만 변경됩니다.

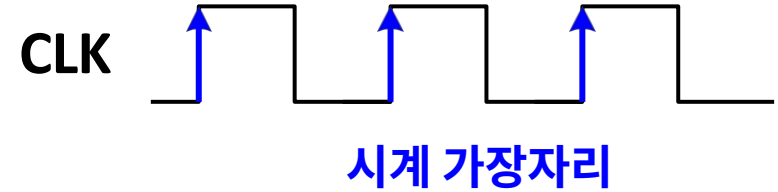
D 플립플롭

기호



- **에지 트리거라고 합니다.**

- 시계 가장자리에서 활성화



# D 플립플롭 내부 회로

- 다음에 의해 제어되는 두 개의 백투백 D 래치(L1 및 L2)  
보완 시계

CLK

- **CLK = 0**인 경우

- L1은 투명
- L2가 불투명합니다.
- D는 **N1**로 통과

D

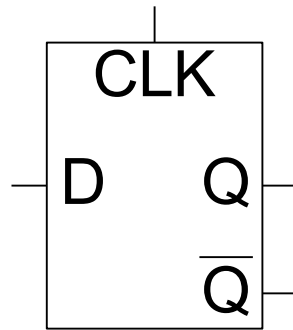
- **CLK = 1**일 때

- L2는 투명합니다.
- L1이 불투명합니다.
- N1은 **Q로 전달됩니다.**

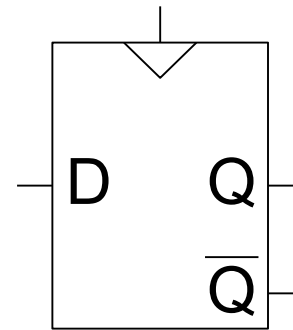
- 따라서 시계의 가장자리에서 (CLK가 0 → 1로 상승할 때)
  - D는 **Q로** 전달됩니다.

# D 래치 대 D 플립플롭

레벨 민감,  
CLK=1 시 투명



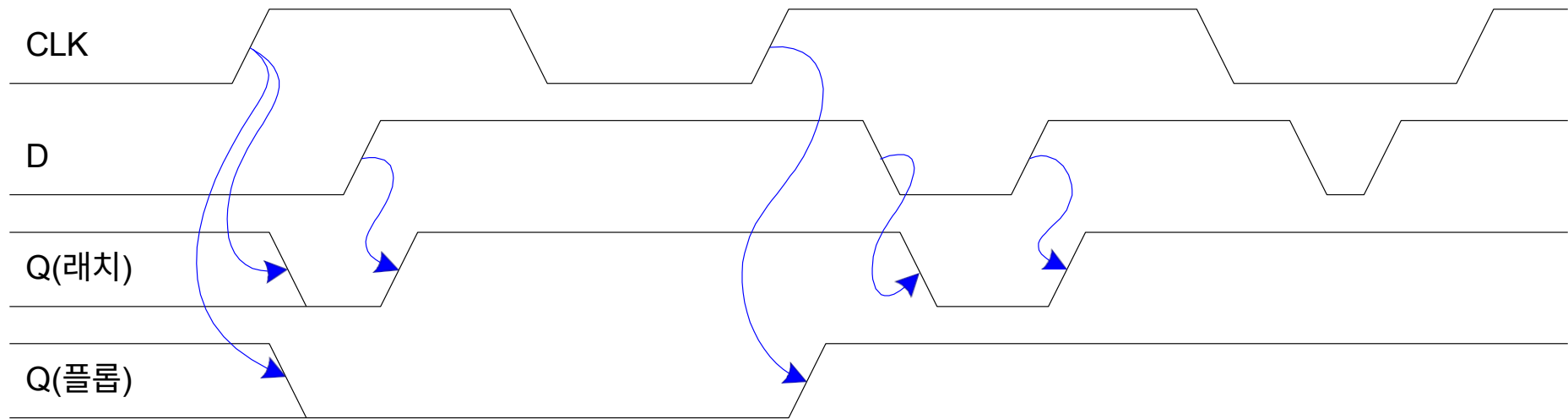
D  
플립플롭



래치D 플

가장자리 트리거  
, 상승 가장자리  
에서 D를 Q로 복  
사합니다.

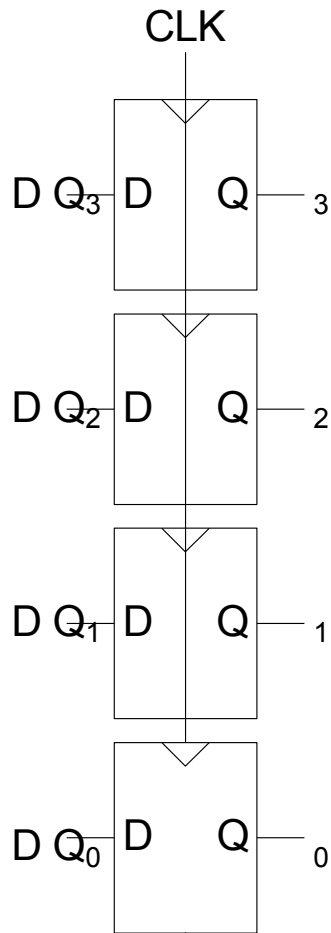




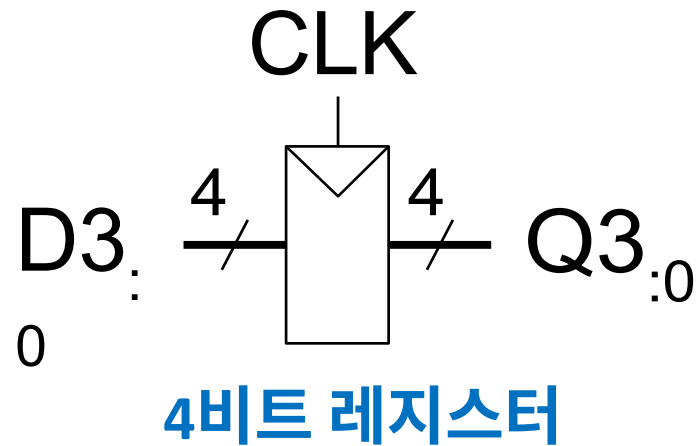
## 3장: 순차 논리

# 플롭의 변형

# 레지스터: 하나 이상의 플립플롭



4비트 레지스터



터

# 레지스터를 그리 는 두 가지 방법

# 플립플롭 활성화

- 입력:  $CLK$ ,  $D$ ,  $EN$

- 활성화 입력( $EN$ )은 새 데이터( $D$ )가 저장되는 시기를 제어합니다.

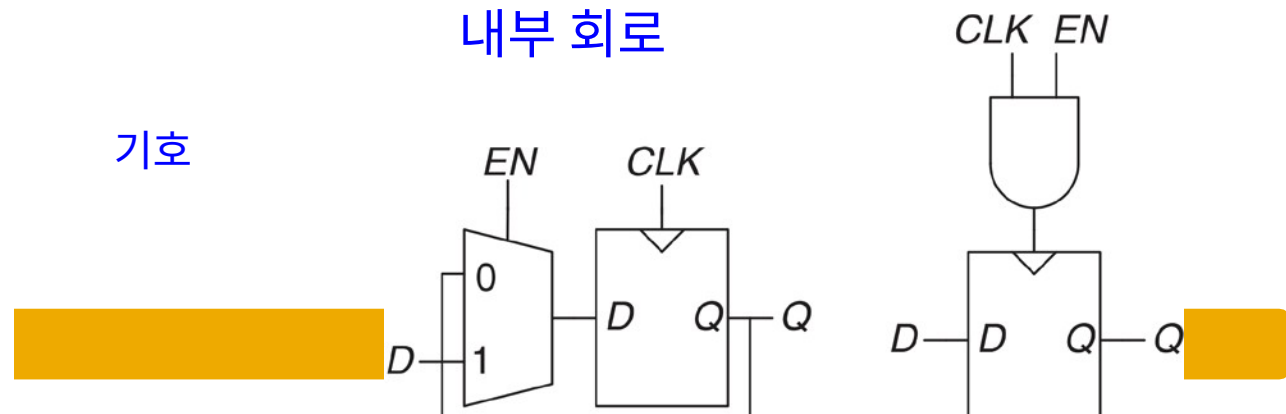
- 기능

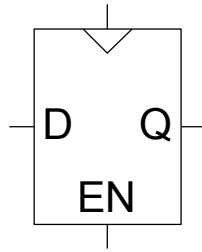
- $EN = 1$ :  $D$ 가 클록 에지의  $Q$ 로 전달됩니다.

- $EN = 0$ : 플립플롭이 이전 상태를 유지합니다.

기호

내부 회로

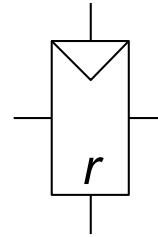
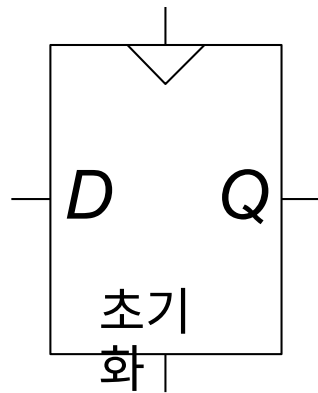




# 리셋 가능한 플립플롭

- 입력:  $CLK$ ,  $D$ , 리셋
- 기능:
  - 초기화 = 1:  $Q$ 가 0으로 강제 설정됩니다.
  - 리셋 = 0: 플립플롭이 일반 D 플립플롭처럼 동작합니다.

기호





# 리셋 가능한 플립플롭

- 두 가지 유형이 있습니다:
  - 동기식: 시계 가장자리에서만 재설정됩니다.
  - 비동기식: 다음과 같은 경우 즉시 재설정됩니다.

초기화 =  
1

내부 회  
로

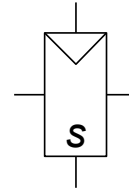
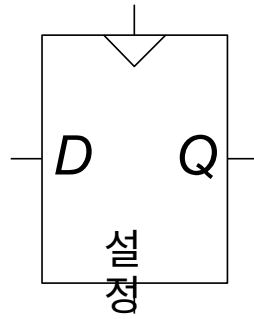
CLK

활성 로우 신호  $\longrightarrow$   $\overline{D}$  초기화 Q

# 설정 가능한 플립플롭

- 입력:  $CLK, D, Set$
- 기능:
  - **설정 = 1**: Q가 1로 설정됩니다.
  - **설정 = 0**: 플립플롭이 일반 D 플립플롭처럼 동작합니다.

기호

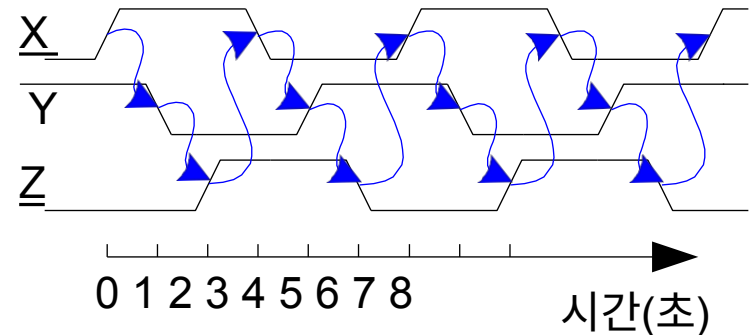
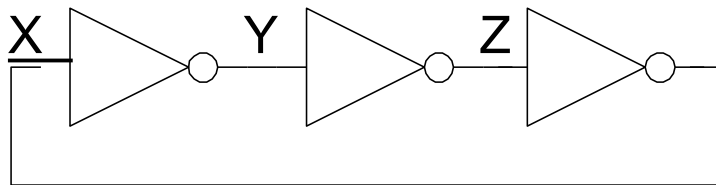


## 3장: 순차 논리

# 동기식 순차 로 직

# 순차 논리

- **순차 회로:** 조합이 아닌 모든 회로를 포함합니다.
- **문제가 있는 회로입니다:**



- **입력 없음** 및 1~3개의 출력
- **불안정한(또는 불안정한) 회로가 진동합니다.**

- 주기는 인버터 지연( $1\text{ns}$ )에 따라 달라집니다.
- **순환 경로가** 있습니다: 출력은 다시 입력으로 피드백됩니다.

# 레이스 조건

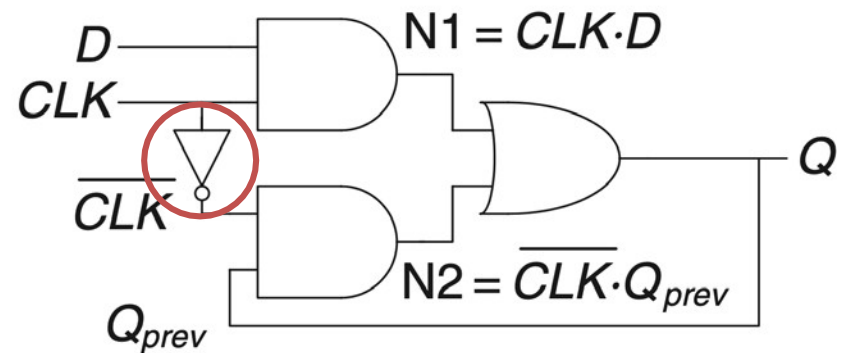
- 개선된(?) D 래치

$CLK$	$D$	$Q_{prev}$	$Q$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

조건

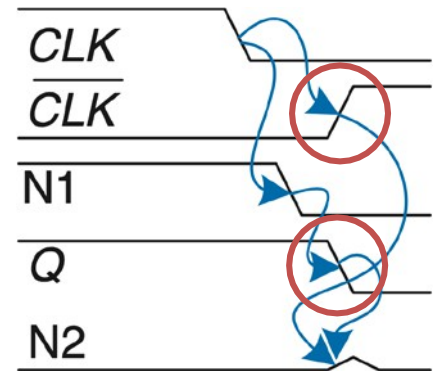
레이스

$$Q = CLK \cdot D + \overline{CLK} \cdot Q_{prev}$$





- D=CLK      됨  
=1,  
Q=1이 - NOT 게이트의 지연이 길면 CLK가  
라고      여전히 0인 상태에서  $Q \rightarrow 0$ ,  $Q_{prev}$   
가정       $\rightarrow 0$ , Q는 영원히 0에 고정됩니다.  
합니  
다.
- CLK  
 $\rightarrow 0$ ,  
N1  $\rightarrow$   
0, N2  
 $\rightarrow 1$   
예상



# 동기식 순차 로직 설계

- 순환 경로는 불안정한 동작을 가질 수 있습니다.
- 레지스터를 **삽입하여** 순환 경로를 끊고 회로를 조합 로직과 레지스터의 모음으로 변환합니다.
  - 레지스터에는 시스템 상태가 포함됩니다.
  - 클럭 에지에서만 상태 변경: 시스템이 클럭에 동기화됨
- 동기식 순차 회로 구성 **규칙**:
  - 모든 회로 요소는 **레지스터** 또는 **조합 회로**입니다.
  - **하나** 이상의 회로 요소는 **레지스터**입니다.
  - 모든 레지스터는 **동일한 시계**를 수신합니다.

- 모든 **순환** 경로에는 적어도 **하나의 레지스터**가 포함됩니다.
- 두 가지 일반적인 동기식 순차 회로
  - 유한 상태 머신(FSM)
  - 파이프라인

# 3장: 순차 논리

**FSM:**

**유한 상태 머신**

# 유한 상태 머신(FSM)

- 다음으로 구성됩니다

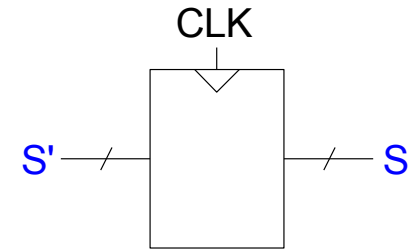
:

## - 상태 등록

- 현재 상태 저장
- 시계 가장자리에서 다음 상태로 로드

## - 조합 논리

- 다음 상태를 계산합니다.



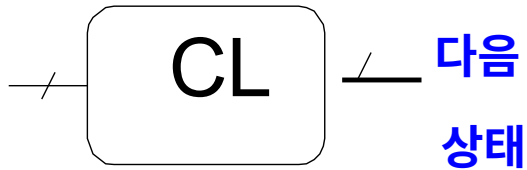
출  
력  
을  
계  
산  
합  
니

다.

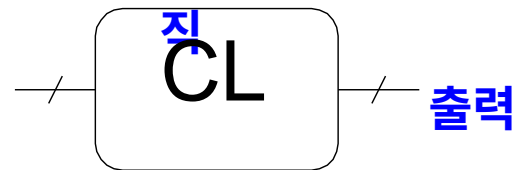
다음  
상태

현재 상  
태

다음 상태 로직



출력 로

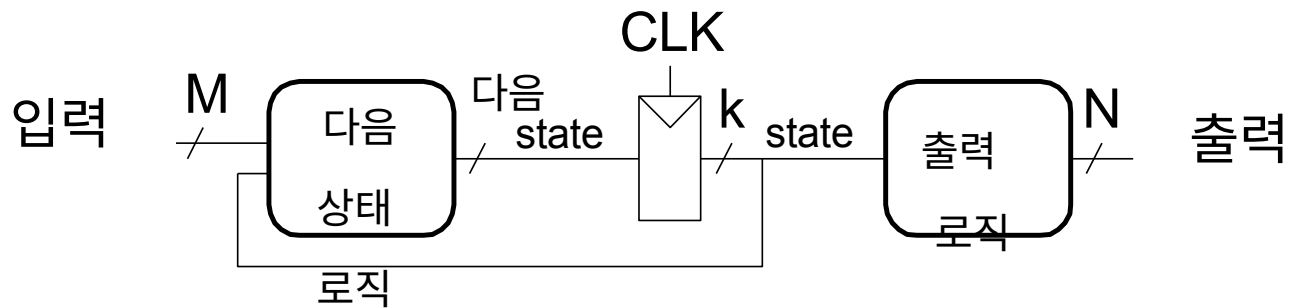


# 유한 상태 머신(FSM)

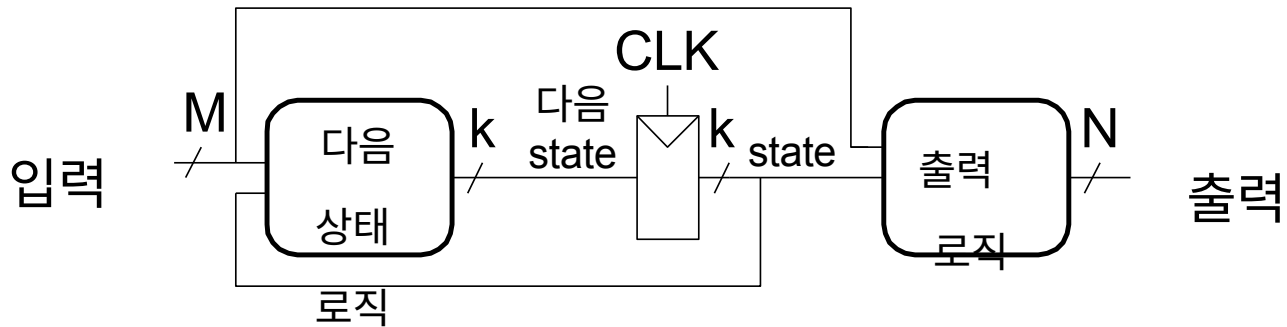
- **현재 상태에 따라 결정되는 다음 상태와 입력**
- 유한 상태 머신의 두 가지 유형은 다음과 같은 점에서 다릅니다.  
**출력 로직:**
  - **무어 FSM:** 출력은 현재 상태에만 의존합니다.
  - **Mealy FSM:** 출력은 현재 상태에 따라 달라집니다.  
**및 입력**

# 유한 상태 머신(FSM)

## 무어 FSM



## Mealy FSM





# FSM 설계 절차

1. **입력 및 출력 식별**
2. **상태 전환 다이어그램 스케치**
3. **상태 전환 테이블 및 출력 테이블 작성**
  - **Moore FSM:** 별도의 테이블 작성
  - **Mealy FSM:** 결합된 상태 전환 및 출력 테이블 작성
4. 이진 형식으로 표현되는 **상태 인코딩**을 선택합니다.
5. 상태 **인코딩**으로 상태 전환 테이블 및 출력 테이블  
다시 쓰기

6. 다음 상태 및 출력 로직에 대한 **부울 방정식** 쓰기
7. **회로도** 스케치

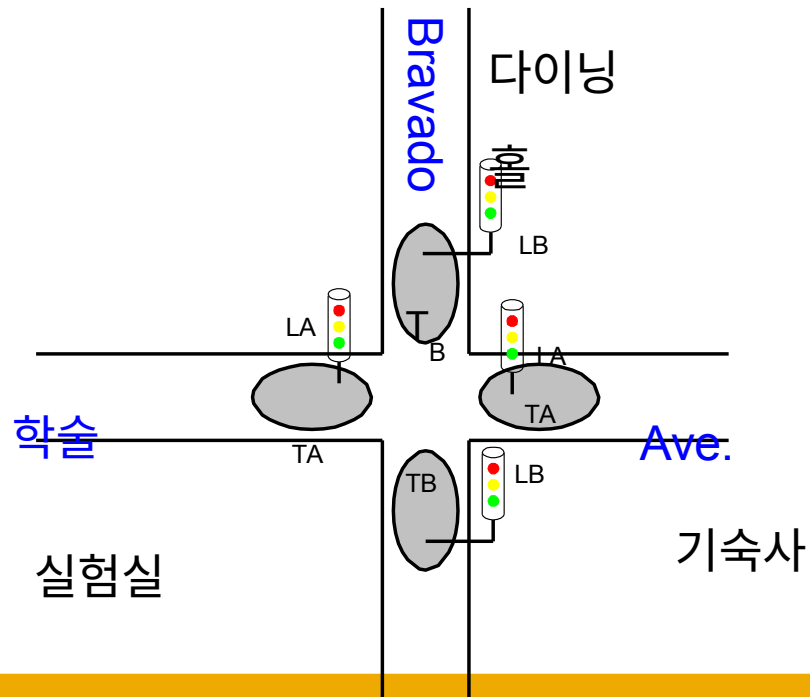
## 3장: 순차 논리

# 무어 FSM 예시

# FSM 예제

- 신호등 컨트롤러

- 교통 센서:  $TA, TB$  (트래픽이 있을 때 TRUE)
- 조명  $LA, LB$

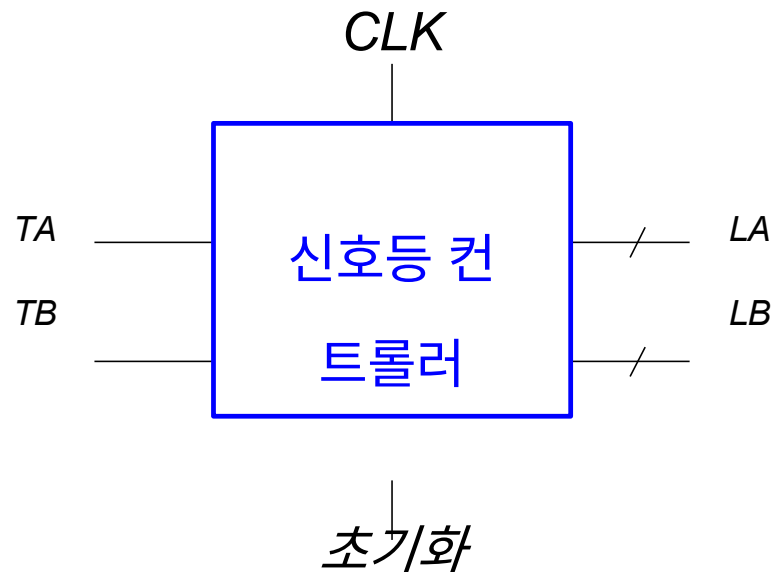


B|v|d.

블  
리

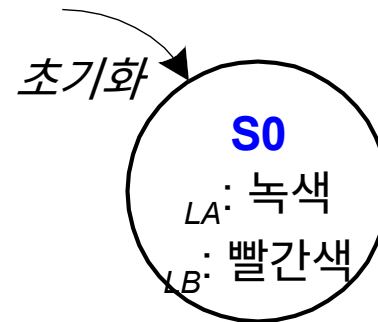
# FSM 블랙박스

- 입력:  $CLK$ , 리셋,  $TA$ ,  $TB$
- 출력  $LA$ ,  $LB$



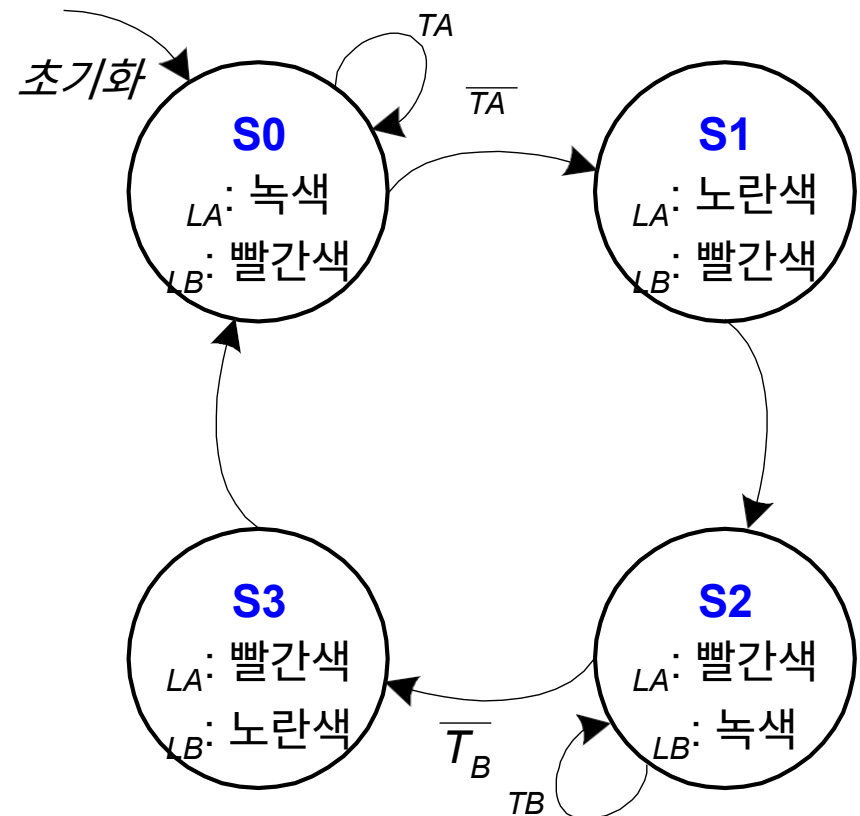
# FSM 상태 전환 다이어그램

- **무어 FSM:** 각 상태에 따라 레이블이 지정된 출력
- **상태:** 상태: 원
- **전환:** 호



# FSM 상태 전환 다이어그램

- 무어 FSM: 각 상태에 따라 레이블이 지정된 출력
- 상태: 상태: 원
- 전환: 호

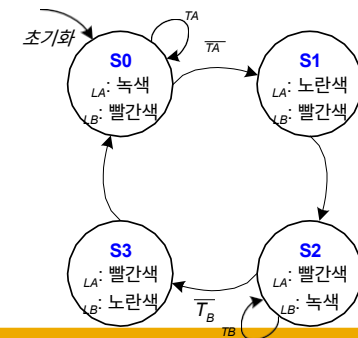




# FSM 상태 전환 테이블

현재 상태 $S$	입력		다음 상태 $S'$
	$TA$	$TB$	
$S_0$	0	X	$S_0$
$S_0$	1	X	$S_1$
$S_1$	X	X	$S_1$
$S_2$	X	0	$S_2$
$S_2$	X	1	$S_3$
$S_3$	X	X	$S_3$

$S$ : 현재 상태  
 $S'$ : 다음 상태



# FSM 인코딩 상태 전환 테이블

현재 상태		입력		다음 상태	
$s_1$	$s_0$	$TA$	$TB$	$S'_1$	$S'_0$
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

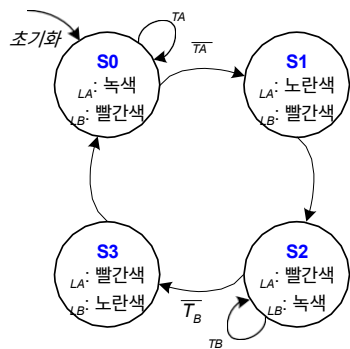
$$S'_1 = \overline{S1}S0 + S1\overline{S0}TB + \overline{S1}S0TB = \overline{S1}S0 + S1S0 = S1 \oplus S0$$

$$S'_0 = S1S0TA + S1S0TB$$

상태	인코딩
S0	00
S1	01
S2	10
S3	11

# FSM 출력 테이블

현재 상태		출력			
S1	S0	LA1	LA0	LB1	LB0
S1	S0	green		red	
S1	S1	yellow		red	
S1	S2	red		green	
S1	S3	red		yellow	낮은



$$L = S S + S S = S$$

$$A1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1$$

$$LA0 = S1S0$$

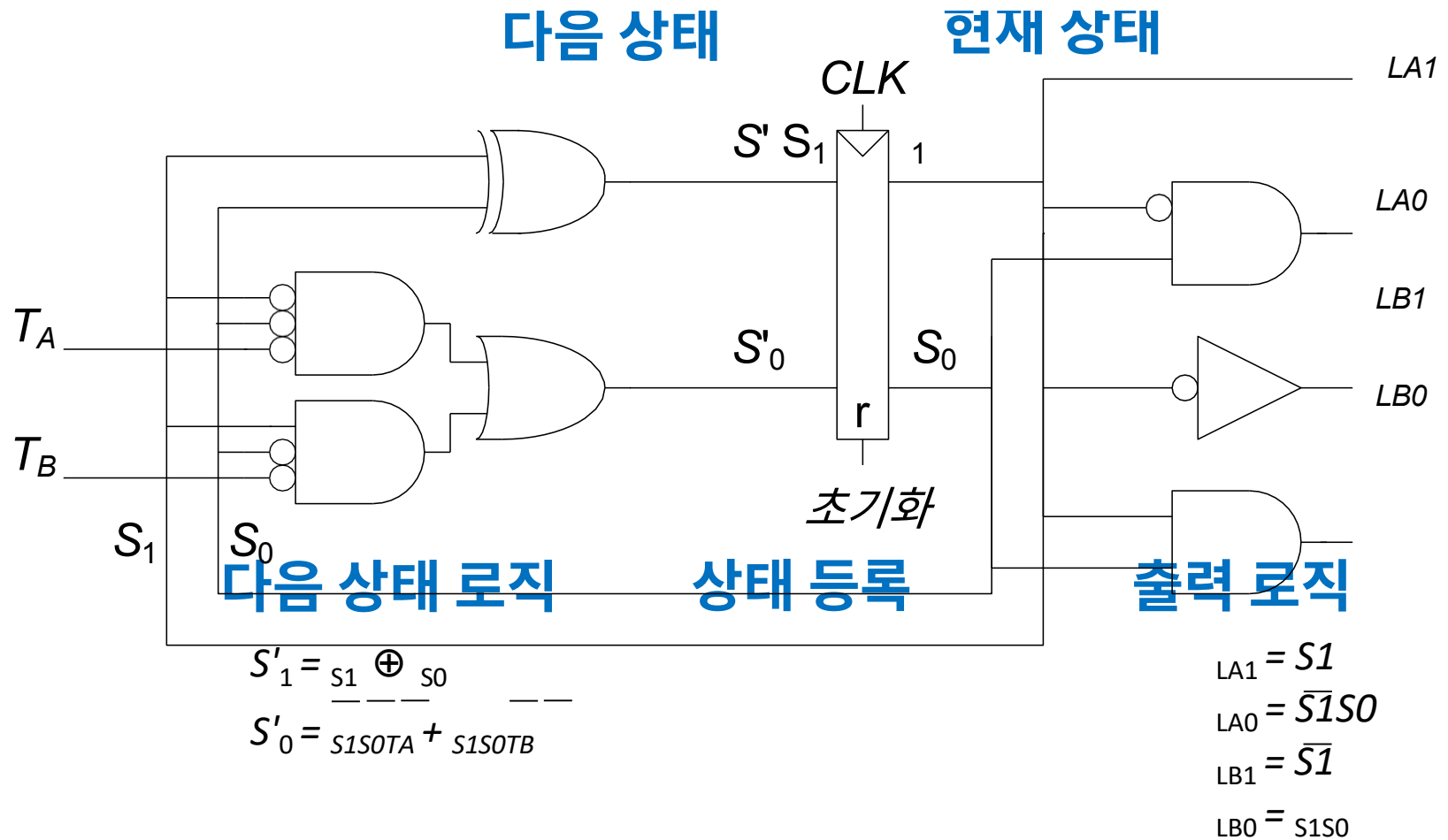
$$LB1 = S1S0 + S1S0 = S1$$

$$L = S S$$

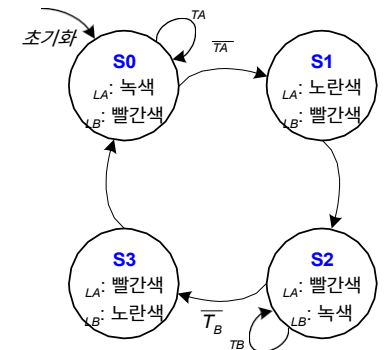
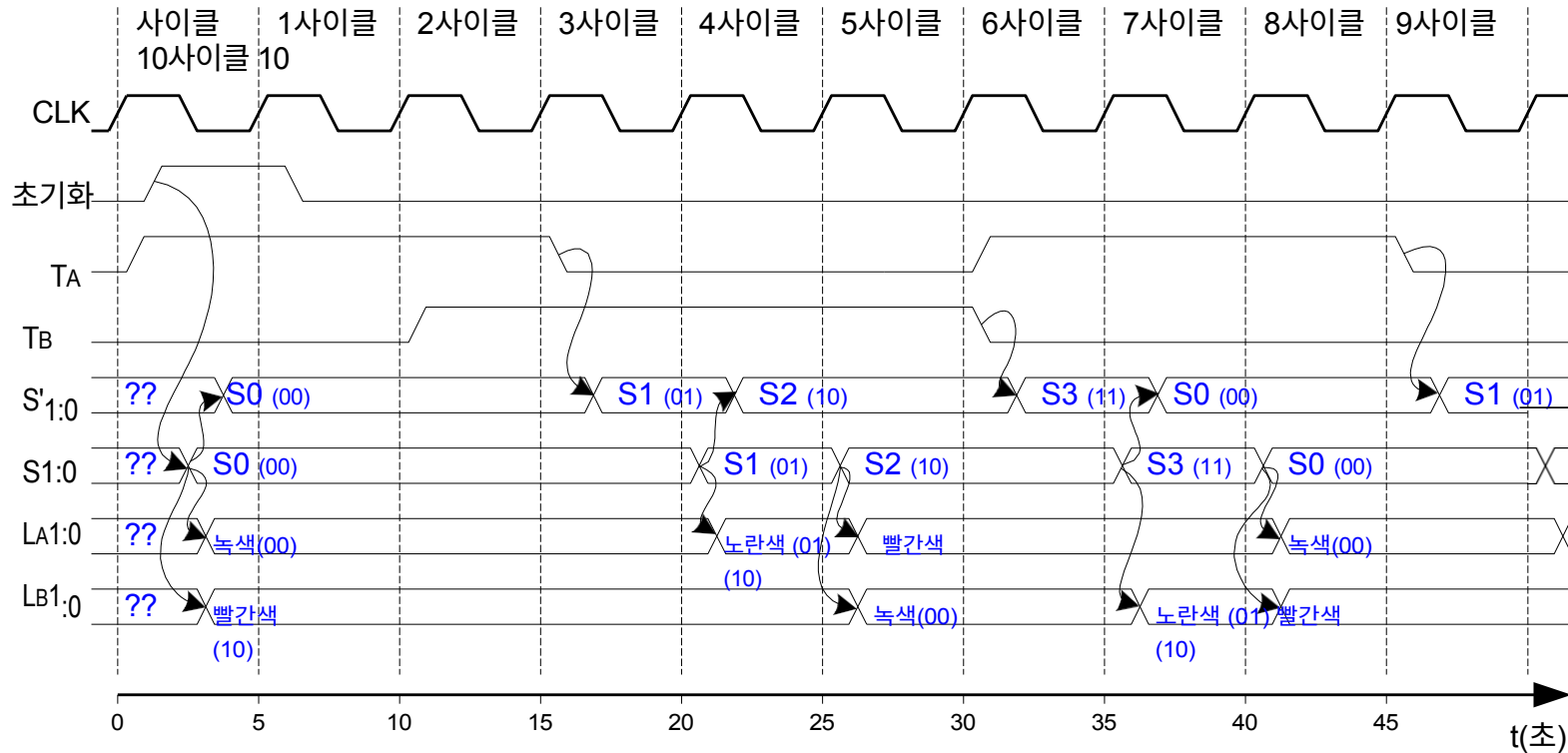
$$B0 \quad 1 \quad 0$$

출력	인코딩
녹색	00
노란색	01
빨간색	10

# FSM 회로도



# FSM 타이밍 다이어그램



# 상태 인코딩

- **이진 인코딩**

- 즉, 00, 01, 10, 11의 네 가지 상태의 경우

- **원핫 인코딩**

- 상태당 하나의 상태 비트
- 한 번에 하나의 상태 비트만 HIGH
- 즉, 4개 주(0001, 0010, 0100, 1000)의 경우

- 플립플롭이 더 필요합니다.
- 다음 상태 및 출력 로직이 더 간단한 경우가 많습니다.

# 1-핫 스테이트 인코딩 예제

0	0	0	1
0	0	0	1
0	0	1	0

0	0	1	0
0	0	0	1
0	1	0	0



현재 상태				입력		다음 상태			
S3	S2	S1	S0	TA	TB	S' <sub>3</sub>	S' <sub>2</sub>	S' <sub>1</sub>	S' <sub>0</sub>
	S0			0	X		S1		
	S0			1	X		S0		
	S1			엑	X		S2		
	S2			<u>스</u>	0		S3		
	S2				1		S2		
	S3			엑	X		S0		

엑

스

엑

스

엑

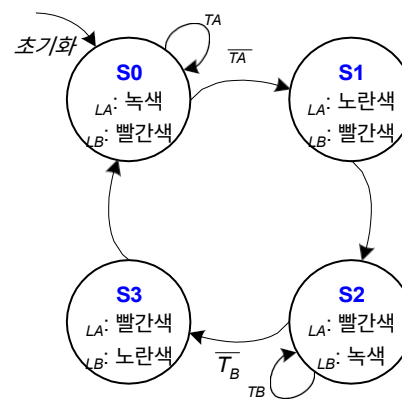
스

$$S' = S \overline{T}$$

$$S_2' = \overset{2}{S} + \overset{1}{S} \overset{2}{T} \overset{B}{}$$

$$S_1' = \underline{SOTA}$$

$$S_0' = \underline{SOTA} + S_3$$



상태	1-핫 인코딩
S0	0001
S1	0010
S2	0100
S3	1000

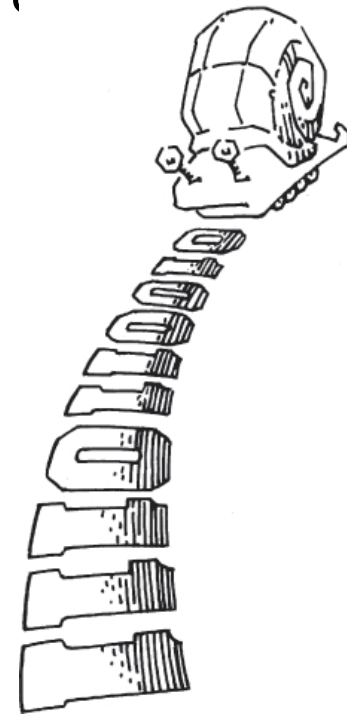
## 3장: 순차 논리

# Mealy FSM 예제

# 무어 대 밀리 FSM

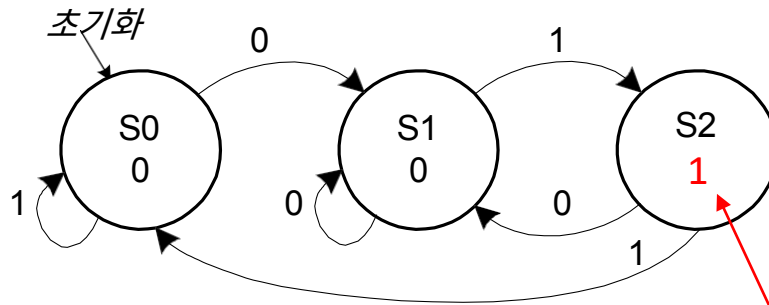
알리사 P. 해커는 1과 0이 적힌 종이 테이프를 기어가는 달팽이를 키우고 있습니다. 달팽이는 기어간 마지막 두 자리가 **01일** 때 마다 미소를 짓습니다. 달팽이가 언제 웃어야 하는지 계산하는 무어와 **밀리의** FSM을 설계합니다.

- **Moore FSM**: 출력은 다음에만 의존합니다.  
현재 상태
- **Mealy FSM**: 출력은 다음에 따라 달라집니다.  
현재 상태 및 입력

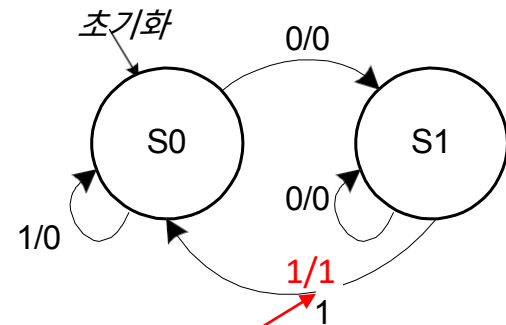


# 상태 전환 다이어그램

무어 FSM



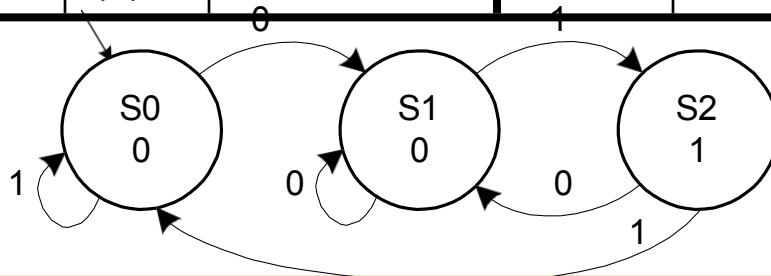
Mealy FSM



출력:  
"달팽이가 웃는다."

# 무어 FSM 상태 전환 표

현재 상태		입력 $A$	다음 상태	
$s_1$	$s_0$		$s'_1$	$s'_0$
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0



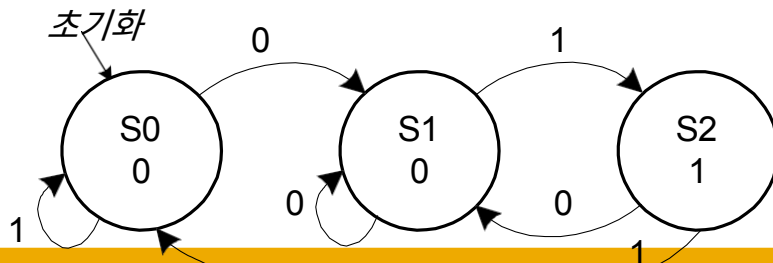
상태	인코딩
S0	00
S1	01
S2	10

$$\begin{matrix} s'_1 & s'_0 \\ s'_0 & s'_1 \end{matrix} = \begin{matrix} s_0 & s_1 \\ s_1 & s_0 \end{matrix} A$$

# 무어 FSM 출력 테이블

현재 상태		출력 Y
S1	S0	
0	0	
0	1	
1	0	

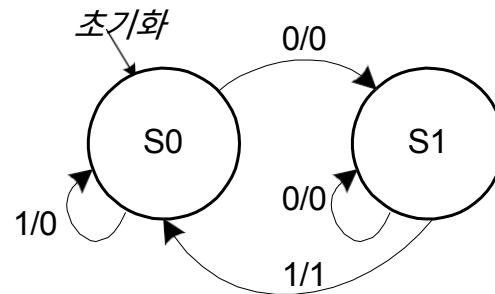
상태	인코딩
S0	00
S1	01
S2	10



# 밀링 상태 전환 및 출력 테이블

현재 상태	입력	다음 상태	출력
$s_0$	$A$	$s'_0$	$Y$
0	0		
0	1		
1	0		
1	1		

상태	인코딩
$s_0$	0
$s_1$	1





# 무어 FSM 회로도

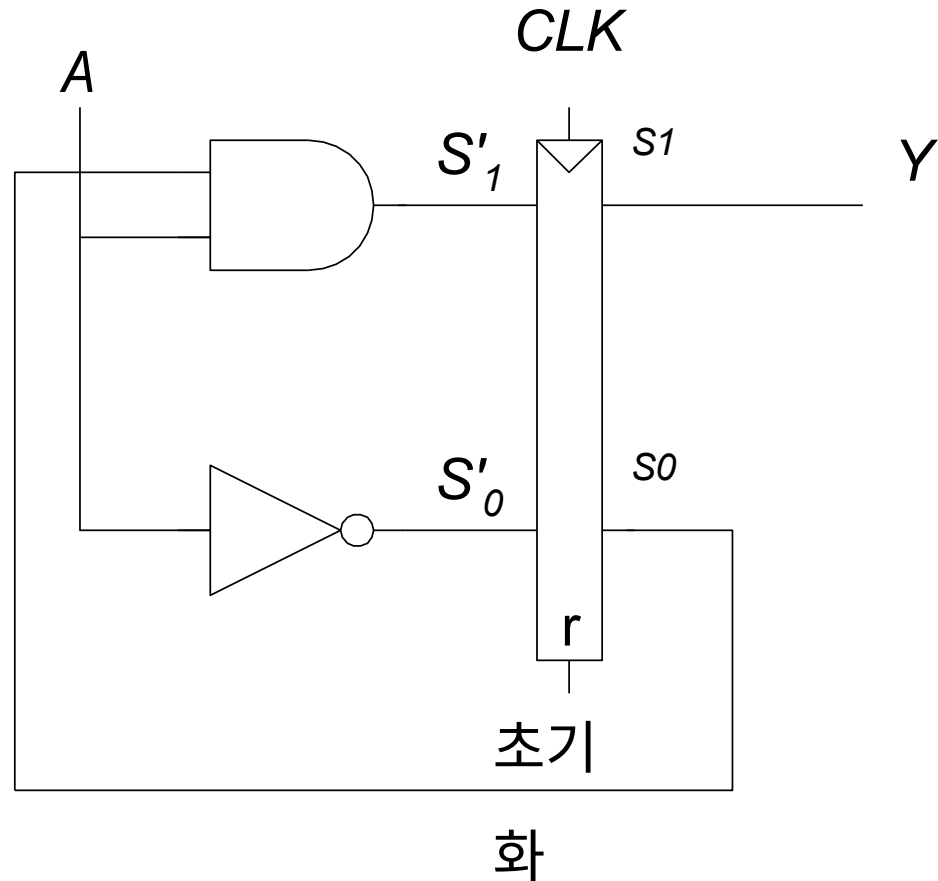
## 다음 상태 방정식

$$S'_1 = S_0 A$$

$$S'_0 = \overline{A}$$

## 출력 방정식

$$Y = s_1$$



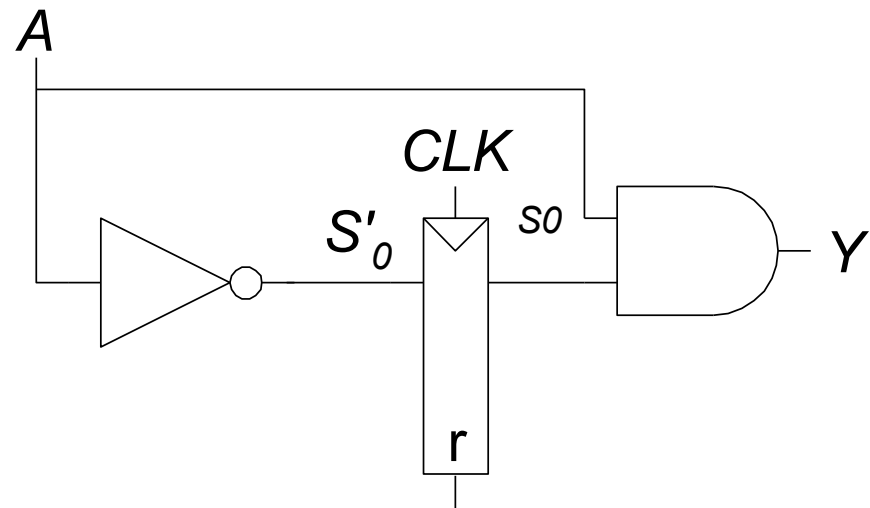
# Mealy FSM 회로도

다음 상태 방정식

$$S^0 = \overline{A}$$

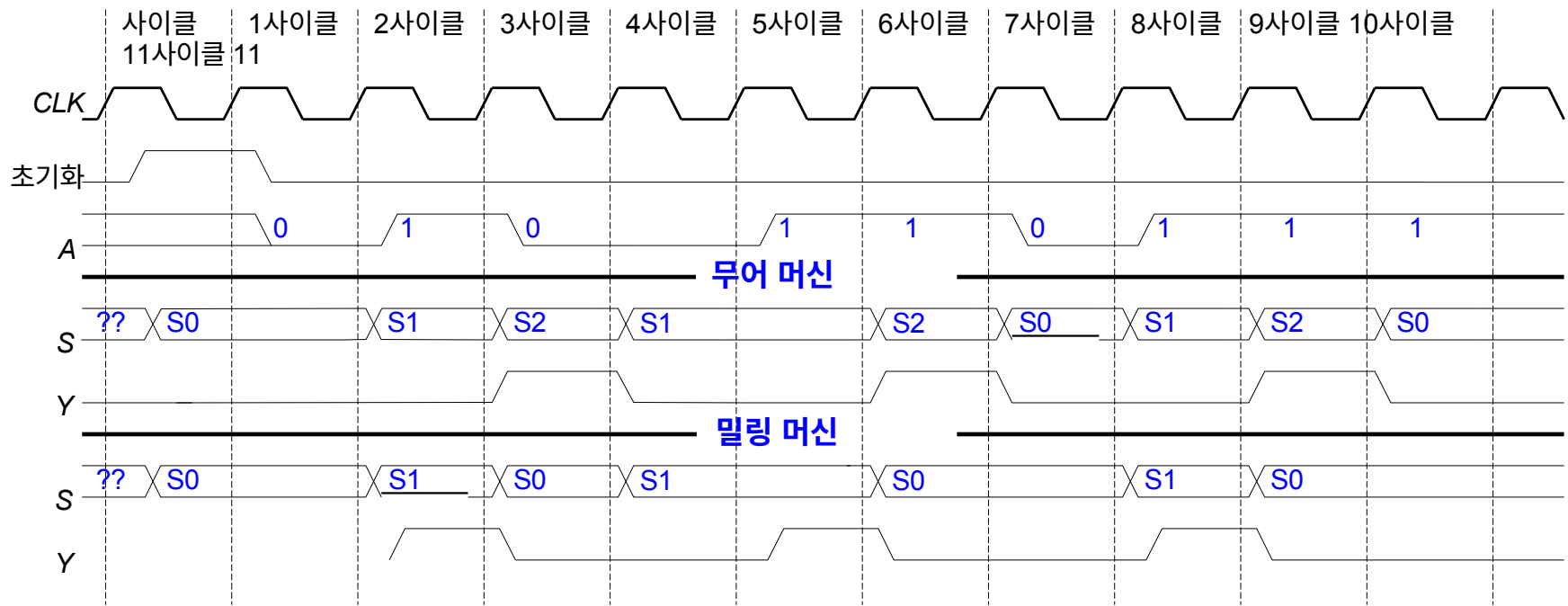
출력 방정식

$$Y = S_0 A$$

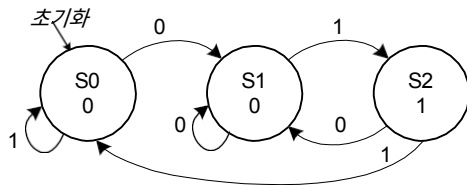


초기화

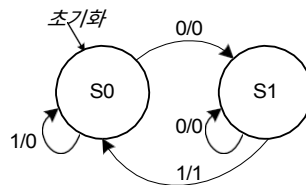
# 무어와 밀리 타이밍 다이어그램



무어 FSM



Mealy FSM



**Mealy FSM:** 입력 패턴 01이 감지되면 즉시 Y를 어설트합니다. **Moore FSM:** 한 사이클 후에 Y를 어설트합니다.

입력 패  
턴 01이  
감지됨

## 3장: 순차 논리

# 타이밍

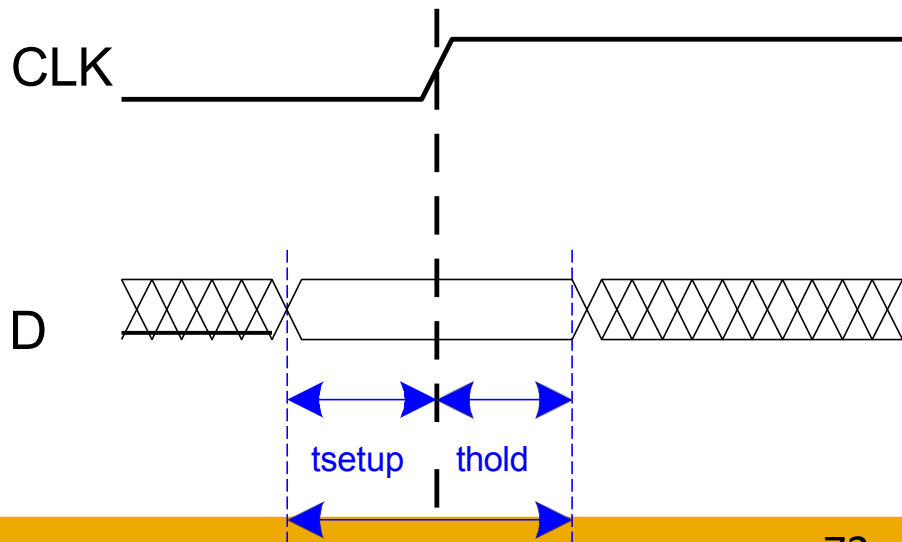
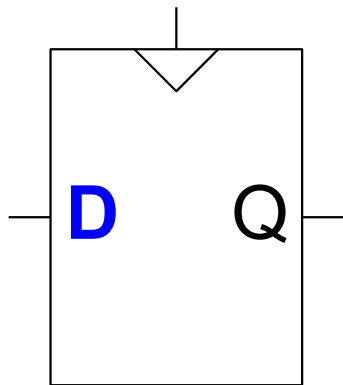
# 타이밍

- 클럭 에지의 플립플롭 샘플  $D$
- **$D$ 는 샘플링 시 안정적이어야 합니다.**
- 사진과 마찬가지로  $D$ 는 시계 가장자리 주변에서 안정적이어야 합니다.
  - 사진을 찍는 동안 물체가 움직이는 경우,  
를 입력하면 이미지가 흐릿해진다



# 입력 타이밍 제약 조건

- **설정 시간:**  $t_{\text{setup}}$  = 클럭 에지 데이터가 안정되어야 하는 시간(즉, 변경되지 않아야 하는 시간)
- **홀드 시간:**  $t_{\text{hold}}$  = 클럭 에지 데이터가 안정적이어야 하는 시간
- **조리개 시간:**  $t_a$  = 시계 에지 데이터 주위의 시간( $t_a = t_{\text{setup}} + t_{\text{hold}}$ ) 이 안정적이어야 합니다.

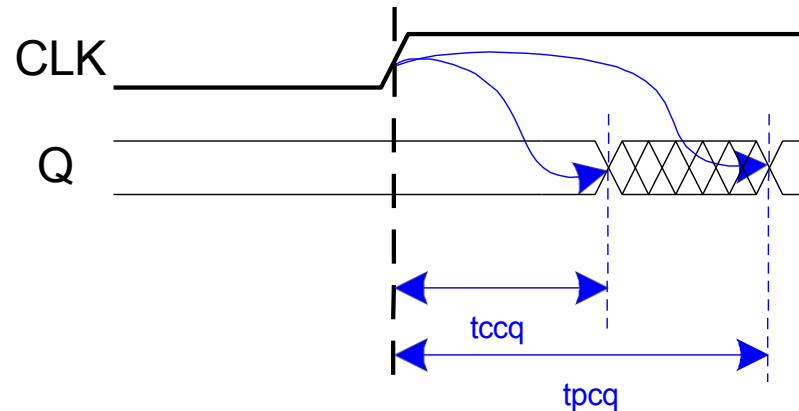
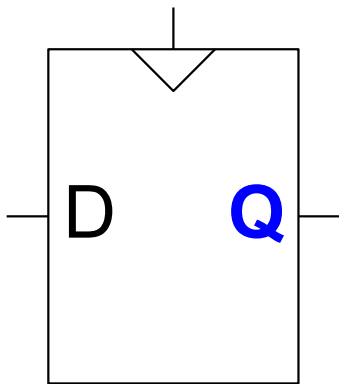






# 출력 타이밍 제약 조건

- **클록 투 Q 전파 지연,  $t_{pcq}$** : 클록 에지 이후 Q가 안정적으로 보장되는 시간(즉, 변경이 중지되는 시간): **최대 지연(클록 투 Q 전파 지연)**
- **클록 투 Q 오염 지연,  $t_{ccq}$** : Q가 불안정해질 수 있는 클록 에지 이후의 시간(즉, 변경 시작): **최소 지연 시간**

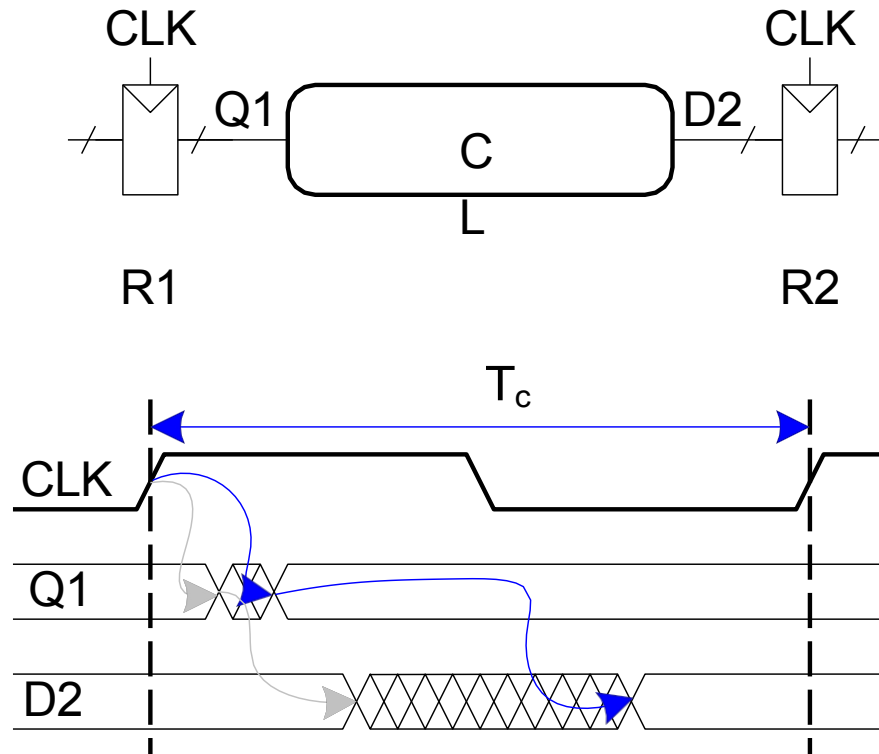


# 동적 규율

- 동기식 순차 회로 입력은 클럭 에지 주변의 **조리개**(설정 및 유지) 시간 **동안 안정적이어야 합니다.**
- 특히 입력은 안정적이어야 합니다.
  - 적어도 시계 가장자리 전에 **설정**
  - 적어도 시계 가장자리 뒤의 **홀드**까지

# 동적 규율

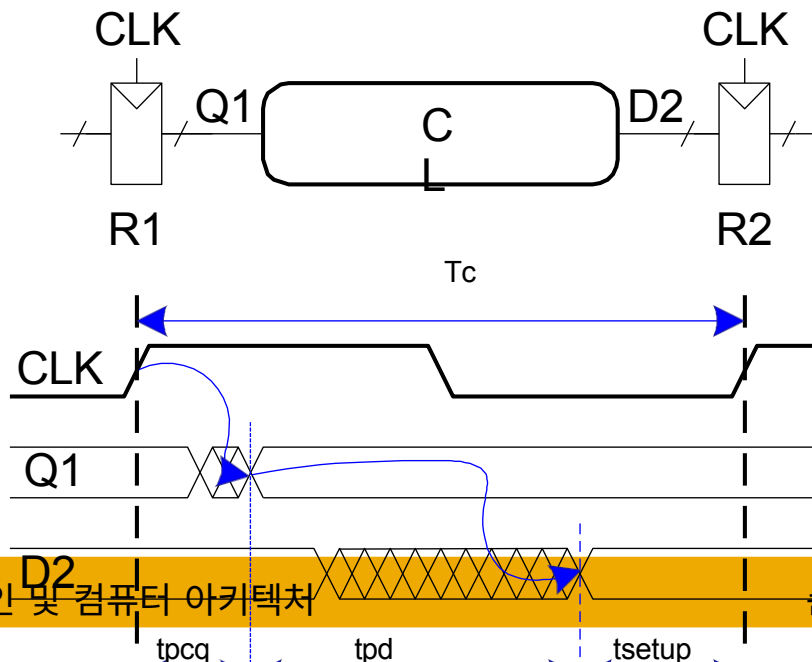
- 레지스터 간 지연은 회로 요소의 지연에 따라 **최소** 및 **최대** 지연이 있습니다.



# 설정 시간 제약

- 조합 로직을 통해 레지스터 R1에서 R2까지의 **최대** 지연에 따라 다릅니다.
  - 레지스터 R2에 대한 입력은 최소한  $t_{\text{setup}}$  이상 안정적이어야 합니다.
- 시계 가장자리 전

라고도 합니다:  
**주기 시간 제약**



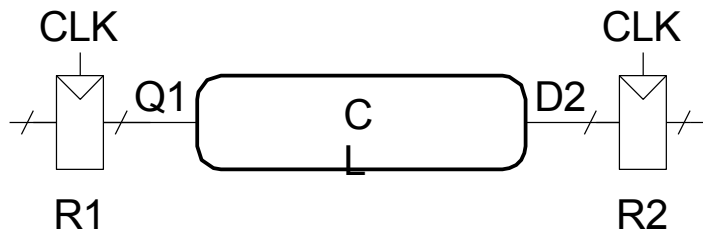
$$T_c \geq t_{\text{pcq}} + t_{\text{pd}} + t_{\text{setup}}$$
$$\leq T_c - (t_{\text{pcq}} + t_{\text{setup}})$$

$$(t_{\text{pcq}} + t_{\text{setup}}):$$

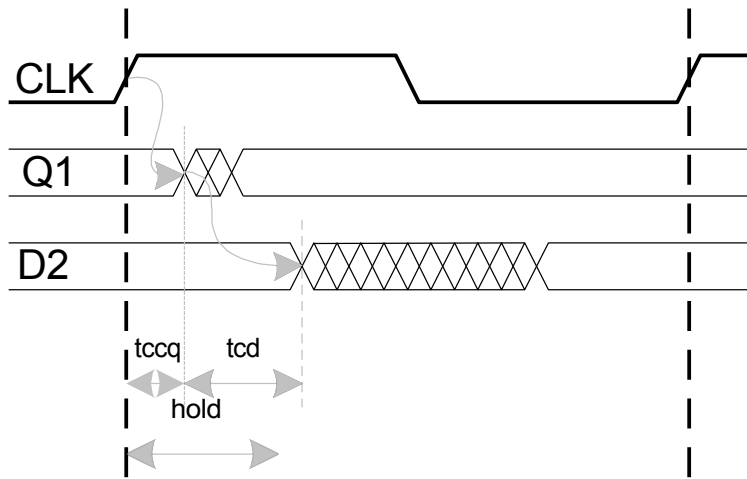
# 시퀀싱 오버헤드

# 보류 시간 제약

- 레지스터 R1에서 조합 로직을 거쳐 R2까지의 **최소** 지연에 따라 다릅니다.
  - 레지스터 R2에 대한 입력은 최소 **홀드** 동안 안정적이어야 합니다.
- 시계 가장자리 뒤



$$\begin{aligned} THOLD &< TCCQ \\ + TCD &> \\ THOLD &\geq TCCQ \end{aligned}$$

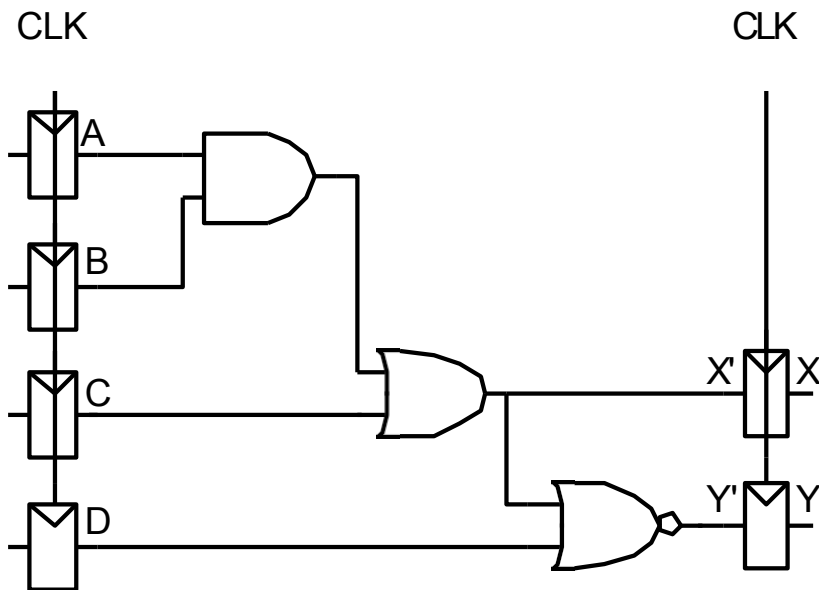


# 타이밍 분석

- **두 제약 조건을 모두** 계산합니다:
  - **설정 시간** 제약(일명 주기 시간 제약)
  - **보류 시간** 제약
- **홀드 시간 제약 조건이 충족되지 않으면 어떤 주파수에서도 회로가 안정적으로 작동하지 않습니다.**



# 타이밍 분석 예시



$$t_{PD} = 3 \times 35 \text{ ps} = 105 \text{ ps}$$

$$t_{cd} = 25 \text{ ps}$$

**설정 시간 제약:**

$$T_c \geq t_{pcq} + t_{pd} + t_{setup_{fc}}$$

## 타이밍 특성

$$\left[ \begin{array}{l} T_{CCQ} = 30\text{PS} \\ T_{PCQ} = 50\text{PS} \\ T_{SETUP} = 60\text{PS} \\ \text{hold} = 70 \text{ ps} \end{array} \right.$$

$$= 1/T_c = 4.65\text{GHz}$$

$$tpd = 35 \text{ ps}$$

$$tcd = 25 \text{ ps}$$

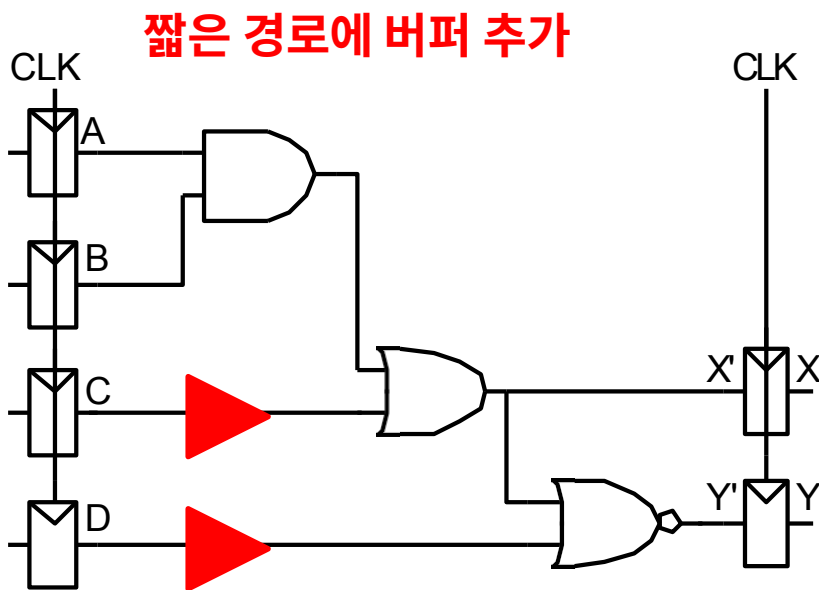
## 시간 제한을 유지합니다:

$$TCCQ + TCD > THOLD ?$$

$$(30 + 25) \text{ ps} > 70 \text{ ps} ? \text{ 아니요!}$$

어떤 주파수에서도 안정적으로 실행되지 않음

# 타이밍 분석 예시



## 타이밍 특성

순회 지연

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$\text{hold} = 70 \text{ ps}$$

$$f_c = 1/T_c = 4.65 \text{ GHz}$$

$$t_{PD} = 3 \times 35 \text{ ps} = 105 \text{ ps}$$

$$T_{CD} = 2 \times 25 \text{ ps} = 50 \text{ ps}$$

## 설정 시간 제약:

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$tpd = 35 \text{ ps}$$

$$tcd = 25 \text{ ps}$$

## 시간 제한을 유지합니다:

$$TCCQ + TCD > THOLD ?$$

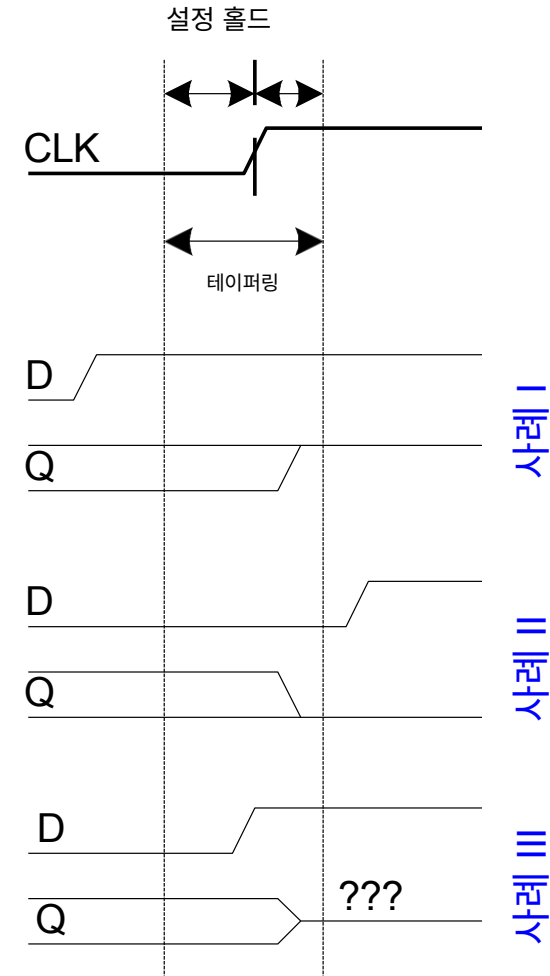
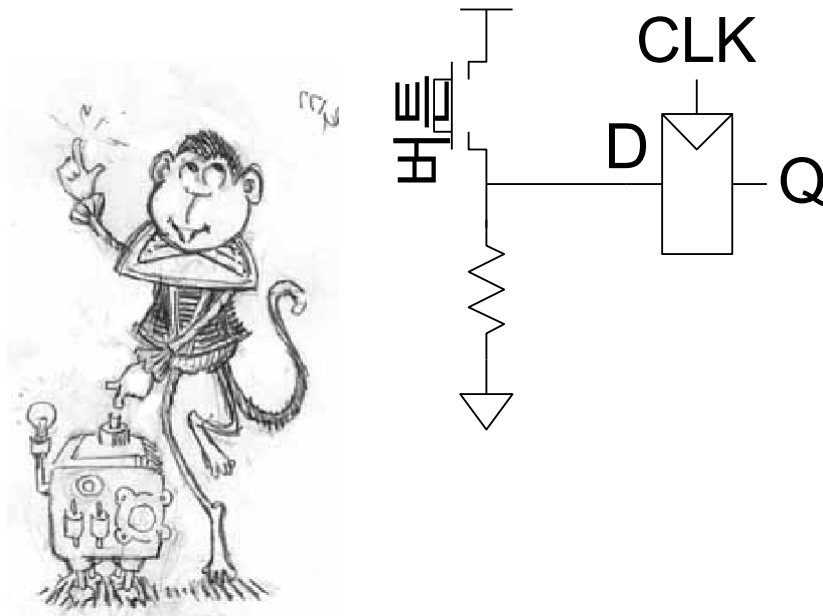
$$(30 + 50) \text{ ps} > 70 \text{ ps} ? \text{예!}$$

## 3장: 순차 논리

# 전이성

# 동적 규율 위반

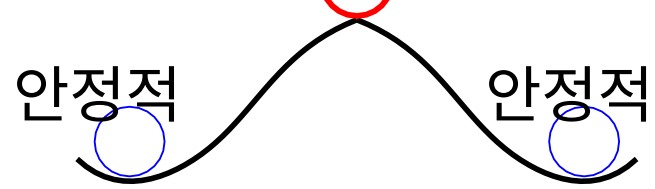
**비동기(예: 사용자) 입력은 동적 규율을 위반할 수 있습니다.**



# 전이성

- **이스테이블 장치:** 두 개의 안정 상태와 그 사이의 준안정 상태
- **플립플롭:** 두 개의 안정 상태(1과 0)와 하나의 준안정 상태
- 플립플롭이 준안정 상태에 놓이면, 일정 시간 동안 그 상태가 유지될 수 있습니다.

## 메타스테이블





## 3장: 순차 논리

# 병렬 처리

# 병렬 처리

- 병렬 처리에는 두 가지 유형이 있습니다:

- 공간 병렬 처리

- 중복 하드웨어는 한 번에 여러 작업을 수행합니다.

- 시간적 병렬 처리

- 작업은 여러 단계로 나뉩니다.
    - 파이프라이닝이라고도 합니다.
    - 예를 들어, 조립 라인

# 병렬 처리

- **토큰:** 출력 그룹을 생성하기 위해 처리된 입력 그룹
- **지연 시간:** 하나의 토큰이 시작부터 끝까지 통과하는 데 걸리는 시간
- **처리량:** 단위 시간당 생성되는 토큰 수

# 병렬 처리로 처리량 증가

# 병렬 처리 예제

- 신호등 제어기 설치를 축하하기 위해 쿠키를 굽는 벤비트디들
  - 쿠키를 굽리는 데 5분
  - 굽는 데 15분
- 병렬 처리 없이 지연 시간과 처리량은 어떻게 되나요?

지연 시간 = 5 + 15 = 20분 = 1/3시간 처리량 =

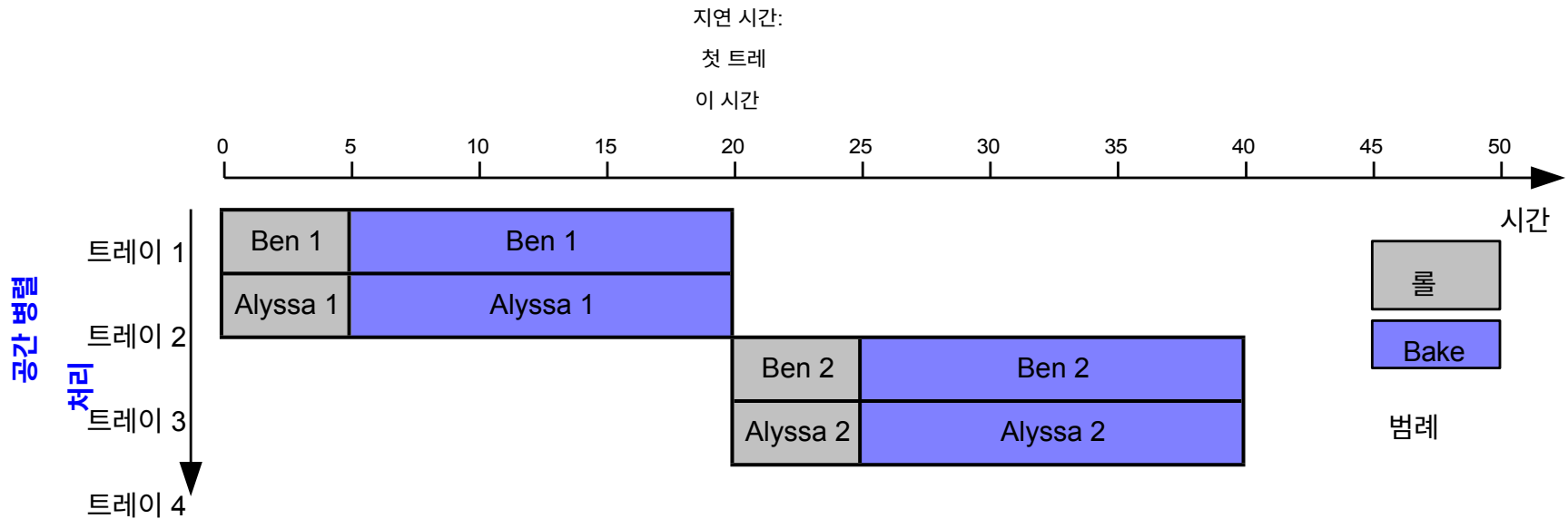
1 트레이 / 1/3시간 = 3 트레이/시간

# 병렬 처리 예제

- Ben이 병렬 처리를 사용하는 경우 지연 시간과 처리량은 어떻게 되나요?
  - **공간 병행성:** 벤은 자신의 오븐을 이용해 엘리사 해커에게 도움을 요청합니다.
  - **시간적 병렬 처리:**
    - **두 단계:** 롤링과 베이킹
    - 그는 두 개의 트레이를 사용합니다.

- 첫 번째 배치가 굽는 동안 두 번째 배치 등을 굽립니다.

# 공간 병렬 처리

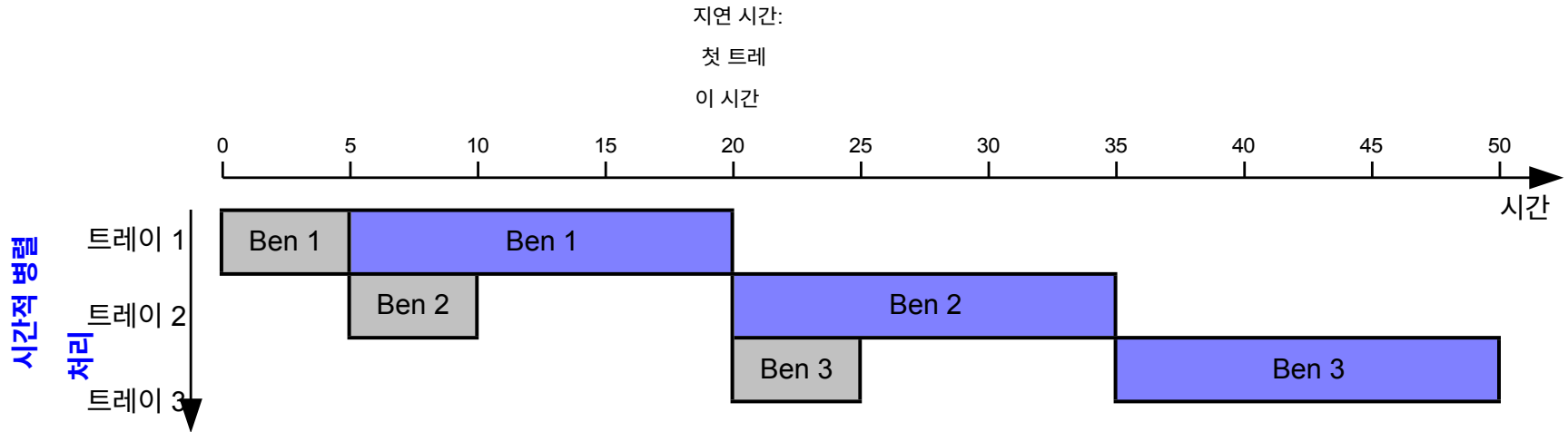


지연 시간 = 5 + 15 = 20분 = **1/3시간** 처리량 =

2 트레이 / 1/3시간 = **6 트레이/시간**



# 시간적 병렬 처리



지연 시간 = 5 + 15 = 20분 = **1/3시간** 처리량 = 1트레

이/4시간 = **4트레이/시간**

두 기술을 모두 사용하면 처리량은 시간당 8트레이가 됩니다.

# 참고 사항

디지털 디자인 및 컴퓨터 아키텍처 강의 노트

© 2021 사라 해리스와 데이비드 해리스

이러한 노트는 출처를 명시하는 한 교육 및/또는 비상업적 목적으로 사용 및 수정할 수 있습니다.