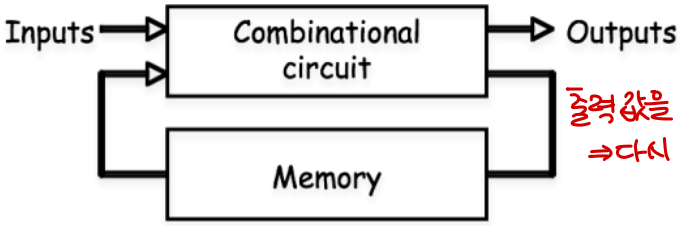
Ch.5 Synchronous Sequential Logic - part A

Sequential circuits



Sequential circuits : the ouput → inputs, state(or the curren contents of some memory)에 영향을 받는 회로 ⇒ 같은 input이더라도 메모리의 상태에 따라 출력이 달라질 수 있음 (같은 입력 → 다른 출력)

↔ **Combinational circuits** : output이 input에만 영향을 받음 (같은 입력 → 같은 출력)

↳ functional programs

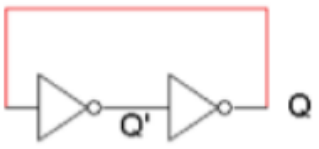
- example : 자물쇠, 엘리베이터, 신호등

enter number ↙ button이 눌렸나? ↳ 현재신호

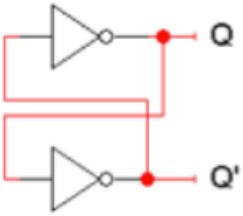
memory

1. hold a value ⇒ data 유지 (change X)
2. read the value that was saved.
3. change the value that's saved.

- **a 1-bit memory**
  - the basic idea of storage → loop
  1. hold a value → 0 or 1
  2. read the bit that was saved
  3. change the value
     ① Set the bit to 1
     ② Reset(clear) the bit to 0

① 

Q or Q' 유지 ⇒ store

② 

Q or Q'을 input으로 ⇒read

↳ prove 하고 싶

③ change는?

change : "read"의 inverter를 NOR gate로 수정

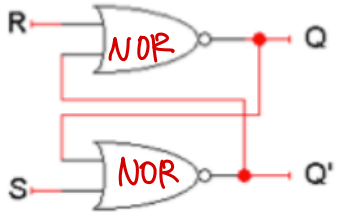
SR latch

S, R 두 가지의 input을 받아 Q와 Q' 두 개의 ouput을 control

→ current output과 next output 시점이 차이 남.

→ **Q next = (R + Q'current)'**
Q'next = (S + Q current)'

⇒ output을 input로 feedback



S	R	Q
0	0	No change
0	1	0 (reset)
1	0	1 (set)

- truth table

Inputs		Current		Next	
S	R	Q	Q'	Q	Q'
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	0	1	0

- state table

→ 현재의 memory도 input에도 영향

① ② ③

- **memory** : set, reset, no change 모두 구현 가능
- **Q** : the data stord in latch → state of the latch ★

현재의 상태값도 input과 고려
⇒ 상태와 출력값이 동일한 경우가 대부분

1. **SR = 00** → no change
   - Q next = (0+ Q'current)' = Q current
   - Q'next = (0 + Q current)' = Q' current

   → ***Qnext = Qcurrent***

   → **hold 구현** (no change)

   (sequential circuits이라는 점을 알 수 있음,

   같은 input이어도 현재 상태에 따라 다른 output)

2. **SR = 10** → set
   - Q'next = (1 + Q current)' = 0
   - Q next = (0+ 0)' = 1

   → ***Qnext = 1, Q'next = 0***

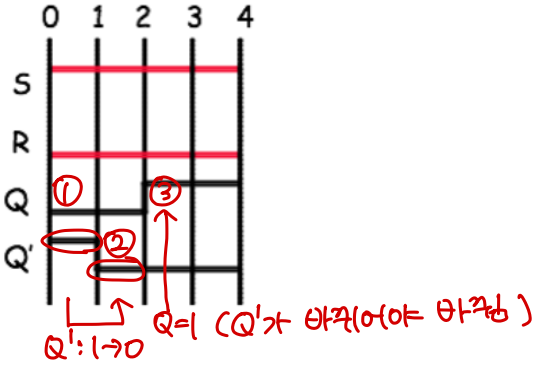
   → **set 구현**

- **two gate delay** 통과해야 정확한 값이 출력 됨

- Qnext가 한 번 1이 되면 output은 바뀌지 않고 계속 1임

⇒ stable state

- latch delay example



⑥ Q=0, Q' = 1 이라고 가정

① S=1 → Q'은 one gate delay 후에 1에서 0으로 바뀜

② R=0 , Q'가 바뀌고 난 후, 즉 two gate delay 후 Q가 1이 됨,

③ 이후 S, R이 바뀔 때까지 stable ⇒ 유지됨

3. **SR = 01** → reset

- Q next = (1+ Q'current)' = 0 (Qcurrent에 상관 x)
- Q'next = (0 + 0)' = 1

→ ***Qnext = 0, Q'next = 1***

→ **reset(clear) 구현**

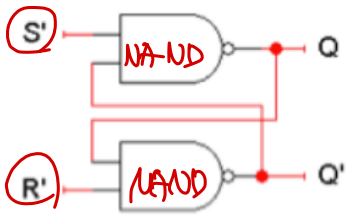
- **two gate delay** 통과해야 정확한 값이 출력 됨

4. **SR = 11**

- Qnext , Q'next 모두 0이 됨 → complement 성립 x ⇒ 다시 1이 각각 들어오면 Qnext와 Q'next 모두 1이 됨
- Q와 Q'가 0과 1을 infinite loop forever.

⇒ **SR=11 input 설정은 worst case!**

S'R' latch



S'	R'	Q
1	1	No change
1	0	0 (reset)
0	1	1 (set)
0	0	Avoid!

$Q_{next} = (S'Q'_{current})'$

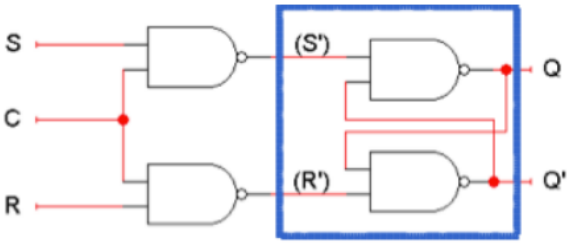
$Q'_{next} = (R'Q_{current})'$

- NOR → NAND gate + input → invert input인 SR latch

⇒ SR latch의 반대 기능 수행

Input		Current		next	
S'	R'	Q	Q'	Q	Q'
1	1	0	1	0	1
1	1	1	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
0	1	0	1	1	0
0	1	1	0	1	0

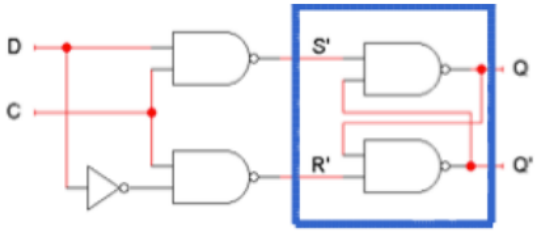
- SR latch with a control input



C	S	R	S'	R'	Q
0	x	x	1	1	No change
1	0	0	1	1	No change
1	0	1	1	0	0 (reset)
1	1	0	0	1	1 (set)
1	1	1	0	0	Avoid!

  - control input C acts like an enable ⇒ 이 들어오면 S,R이 무엇이든 Enable

D latch



C	D	Q
0	x	No change
1	0	0
1	1	1

- C=0 → S,R에 상관 없이 state Q는 no change
- C=1 → input D = state Q ⇒ S'R'이 no change가 되기 때문에 D 그대로 통과

⇒ set, reset input이 따로 없음

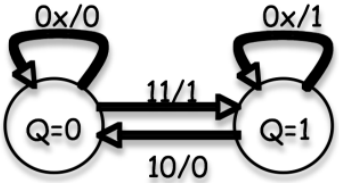
⇒ SR latch와 다르게 bad input이 없음. 모두 vaild

State diagram

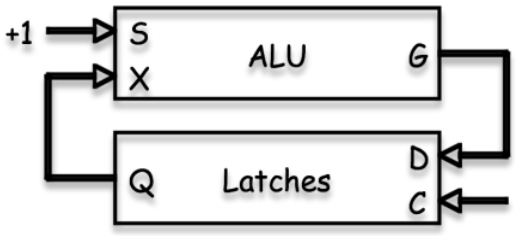
sequential circuits 표현을 위해 state diagram 사용

→ arrow : (input) / (output)

node : (state)



Using latches in real life



- **ALU** : 산술논리연산 장치
  - G = X+1 operation 적용 가능
  - 증가된 value가 latch에 들어 옴 , 연산을 중단하고 싶을 때는 latch를 disable
  - But, disable the latches에서 ALU가 output을 만들 동안 충분히 기다리지 못 함.
    - ALU operation은 dealy 발생 가능(adder gate) , carry도 delay에 영향
    - 연산이 얼마나 오래 걸리는지, latch가 얼마나 enable할 지 알아야 함.