**컴퓨터 구조 프로젝트 #0**

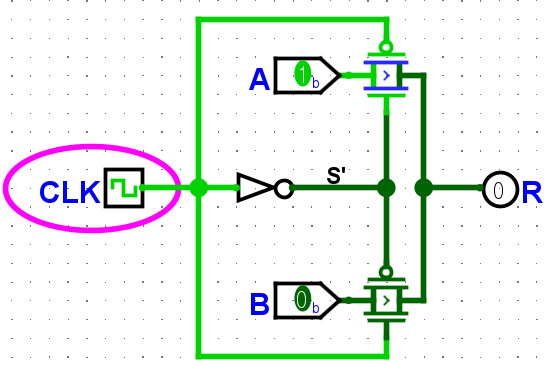
**컴퓨터정보공학부 2018202076 이연걸**

2. Assignment

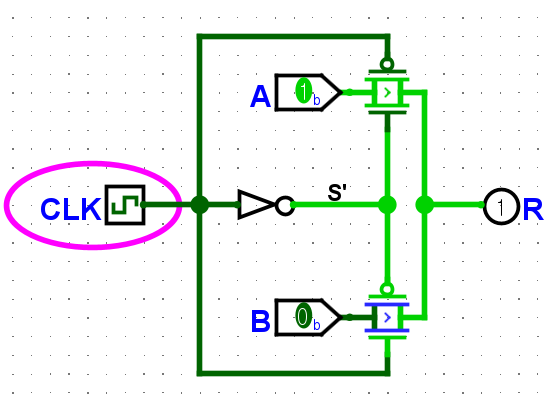
2-1, 1.

Transmission gates란 입력에서 출력으로 신호를 선택적으로 차단하거나 통과시키는 전자 소자이다. nMOS와 pMOS로 구성되어 있으며 양방향 스위치의 역할을 해 analog switch라고 부르기도 한다.

2-1, 2.



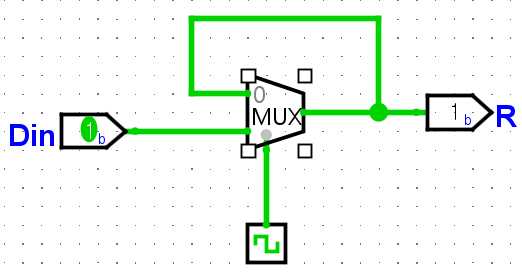


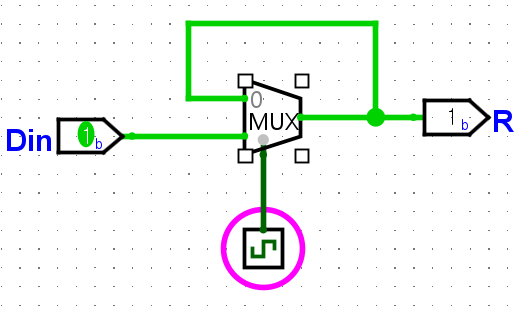


Transmission gate두개와 CLK, NOT gate, Input과 Output pin을 사용해 2:1 MUX를 구현했다. Selection signal S에 의해 출력 값이 입력A가 될 것인지, 입력 B가 될 것인지를 결정한다. 이번 실습에서는 CLK가 high 일 때 입력B가 Transmission gate를 통과하고, CLK가 row일 때 입력A가 Transmission gate를 통과하게 구현했다.

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **CLK** | **R** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

2-1, 3.



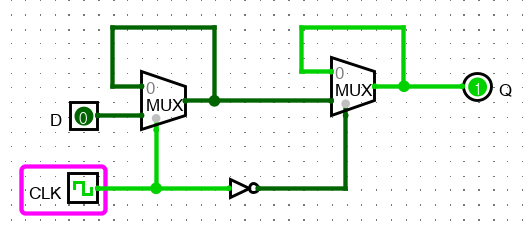


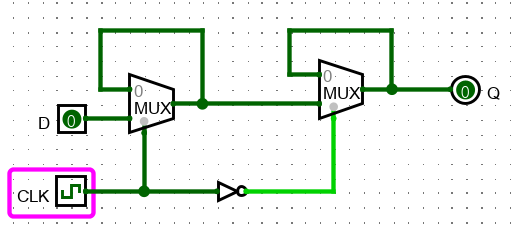
2:1 MUX와 CLK, Input과 Output pin을 사용해 Latch를 구현했다. MUX는 선택한 입력 값(CLK)에 따라 두 입력 중 하나를 선택한다. 지금 실습에서 사용된 두 입력 중 하나(이하 A)는 현재 통과된 신호를 그대로 입력으로 사용한다. A는 이전 신호(Din)를 유지하고 있으므로 CLK가 LOW로 바뀌어도 Din이 아닌 A를 통과시킨다 해도 Din의 값을 갖는다. 따라서 현재의 값을 유지하는 Latch의 기능을 할 수 있다.

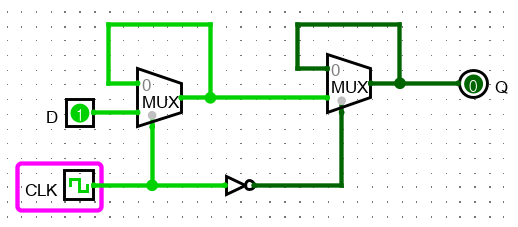
|  |  |  |
| --- | --- | --- |
| **Din** | **CLK** | **R** |
| X | 0 | X |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

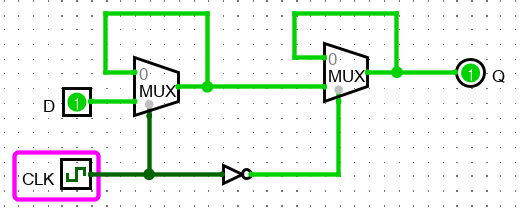
Positive level triggered

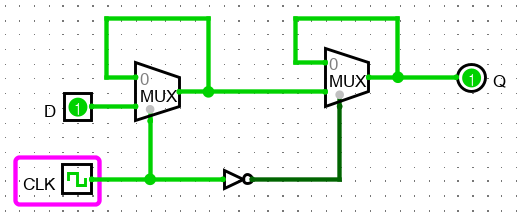
2-1, 4.

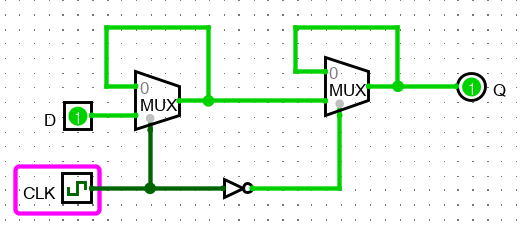


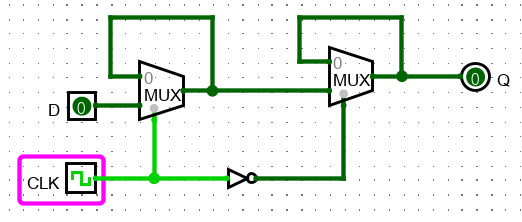


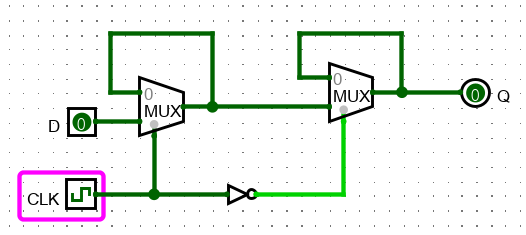








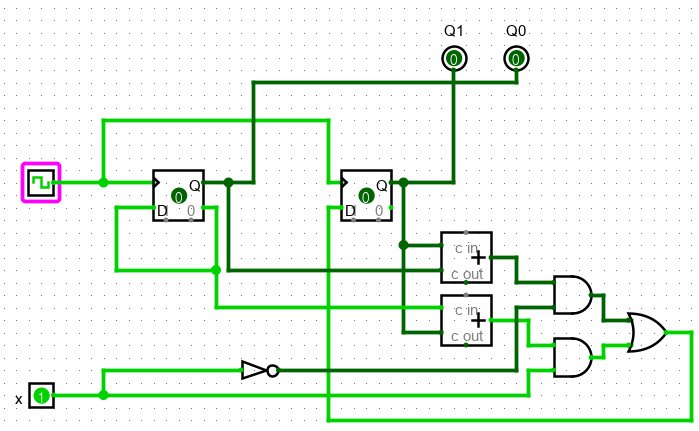


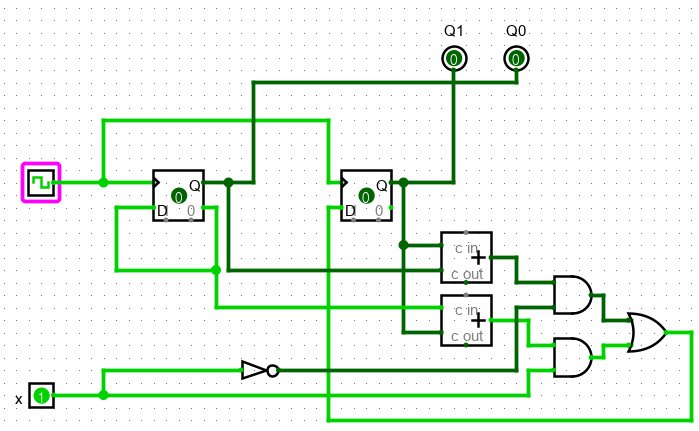


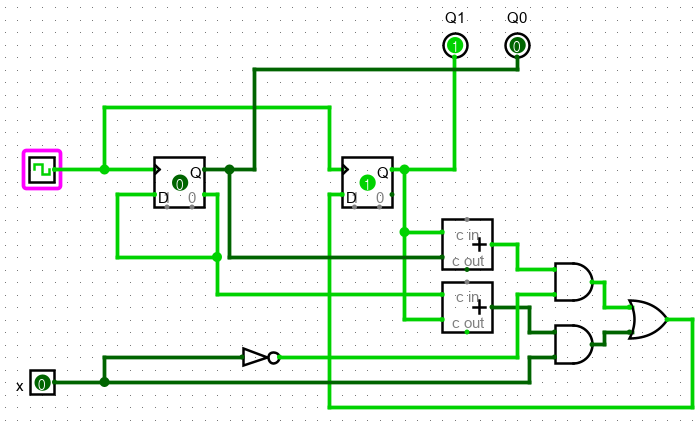
문제 3번에서 만든 Latch 2개와 CLK, Input과 Output pin을 사용하여 D Flip-Flop을 구현했다. 이번 실습에 구현한 D Flip-Flop은 하강 Edge Flip-Flop으로 CLK이 1에서 0으로 바뀌는 구간에 Q의 값이 변한다. 위의 그림을 보면 확인할 수 있듯이 D가 0 Q가 1인 상태에서 하강 Edge가 트리거 되면 Q가 0으로 바뀌고 D가1 Q가 0인 상태에서 하강 Edge가 트리거 되면 Q가 1로 바뀐다. 또한 D가 0 Q가 0 인 상태에서 하강 Edge가 트리거 되면 Q의 변화가 없고 D가 1 Q가 1 인 상태에서 하강 Edge가 트리거 되어도 Q의 변화가 없다. 이처럼 Flip-Flop은 CLK 입력에 반응하여 출력의 상태를 바꾸는 기억소자이다.

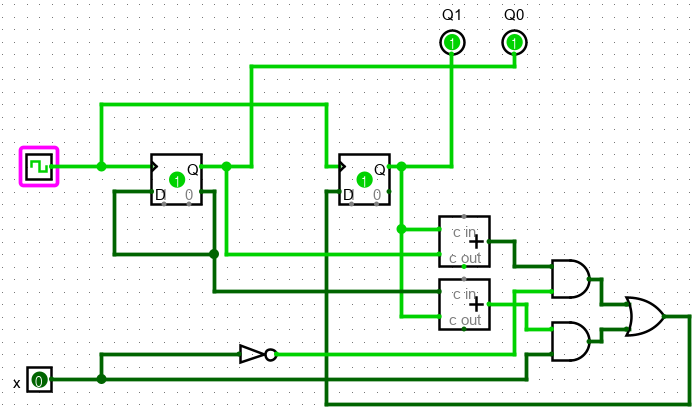
|  |  |  |
| --- | --- | --- |
| D | Q | Q+(다음 하강 Edge때 Q값) |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

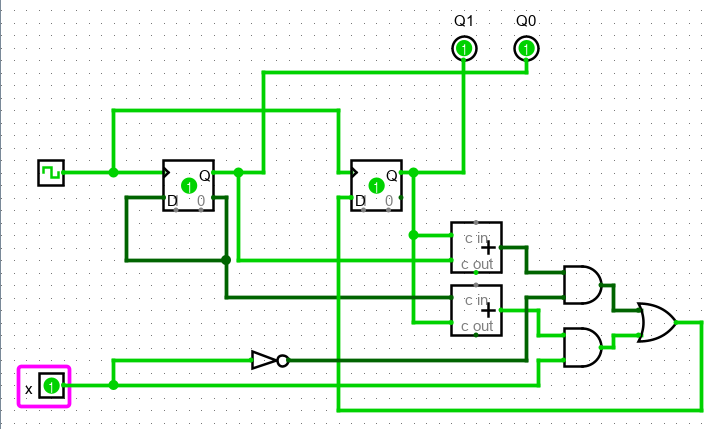
2-1, 5.

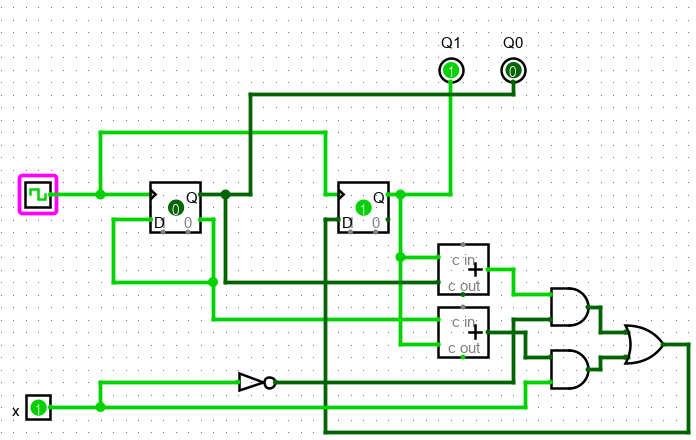


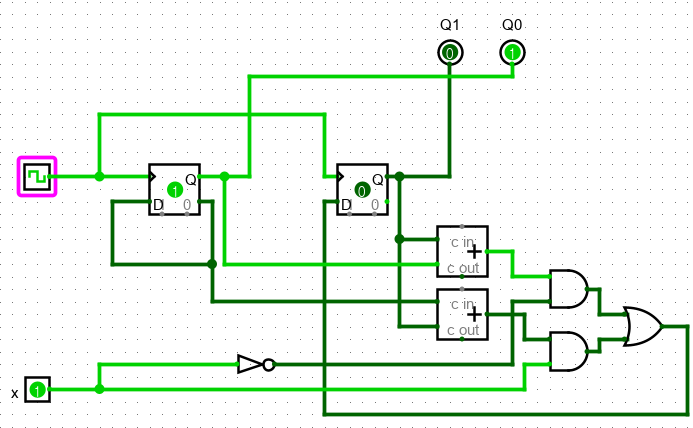


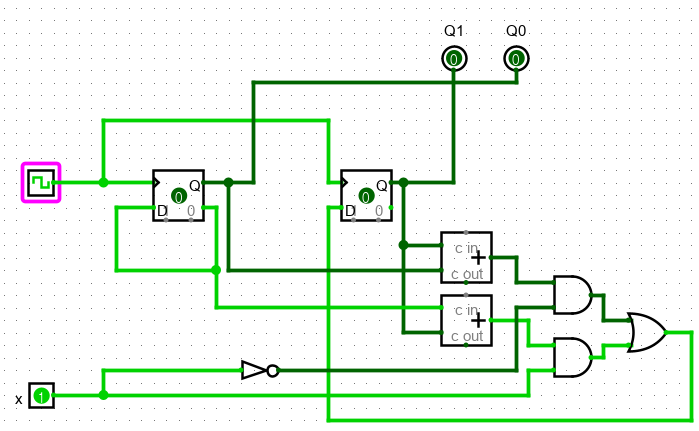












2개의 D Flip-Flop과 2개의 Adder, 2개의 AND Gate, 1개의 OR Gate, 1개의 NOT Gate, CLK 그리고 Input, Output pin을 사용해 2bit UP/DOWN synchronous Counter를 구현하였다. 사용된 Flip-Flop은 rising edge로 CLK이 1로 변할 때 마다 출력 값에 변화가 생긴다. X가 0일 때 순방향(Up counter)이며 x가 1일 때 역방향 (Down counter)이다. Q0와 Q1의 현재 상태가 00, 01, 10, 11이라 한다면 다음 상태는 x=0 : 01 10 11 00, x=1 : 11 00 01 10이 된다. 이를 진리표로 그리면

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **현재상태** | | **다음상태** | | **D플립플롭** | |
| **X** | **Q1** | **Q0** | **Q1** | **Q0** | **D1** | **D0** |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

이렇게 되고 이를 통해 카르노맵을 그려 논리식을 구했다. 이때 논리식은

D1=x’(Q1’Q0 + Q1Q0’) + x(Q1’Q0’ + Q1Q0)

D0=Q0’

이를 토대로 카운터를 구현하였다. 이때 Adder를 사용하는데 Adder의 기능은 1비트의 덧셈일 때 0+0=0, 0+1=1, 1+0=0, 1+1=0 이다. 이것은 XOR게이트의 동작과 흡사하며 이를 사용해 D1 Flip-Flop의 입력을 만들었다. 예를 들어 Q1과 Q0를 Adder에 입력으로 넣고 그 출력을 논리식으로 나타낸다면 Q1’Q0 + Q1Q0’가 된다. 남은 식들은 논리 게이트를 사용해 묶어주었다.

2-1, 6.

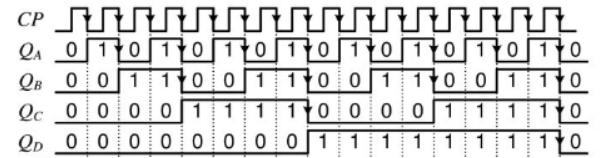
Synchronous Up/Down Counter와 Asynchronous Counter는 Flip-Flop들의 CLOCK이 어디에 연결되어 있는가에 차이가 있다.

동기 상향 하향 카운터는 모든 Flip-Flop들이 하나의 공통 클럭에 연결되어 있어서 모든 Flip-Flop이 동시에 트리거 된다. 그래서 비교적 설계가 복잡하지만 전달지연이 없이 각각의 Flip-Flop들이 즉시 반응한다.

비동기 상향 하향 카운터는 첫 번째 Flip-Flop의 CLOCK에만 클럭 펄스가 입력되고, 다른 Flip-Flop은 이전 Flip-Flop의 출력을 CLOCK의 입력으로 받아 사용된다. 이는 설계를 간단하게 해주지만 뒤의 Flip-Flop으로 갈수록 전달지연 시간이 높아진다.

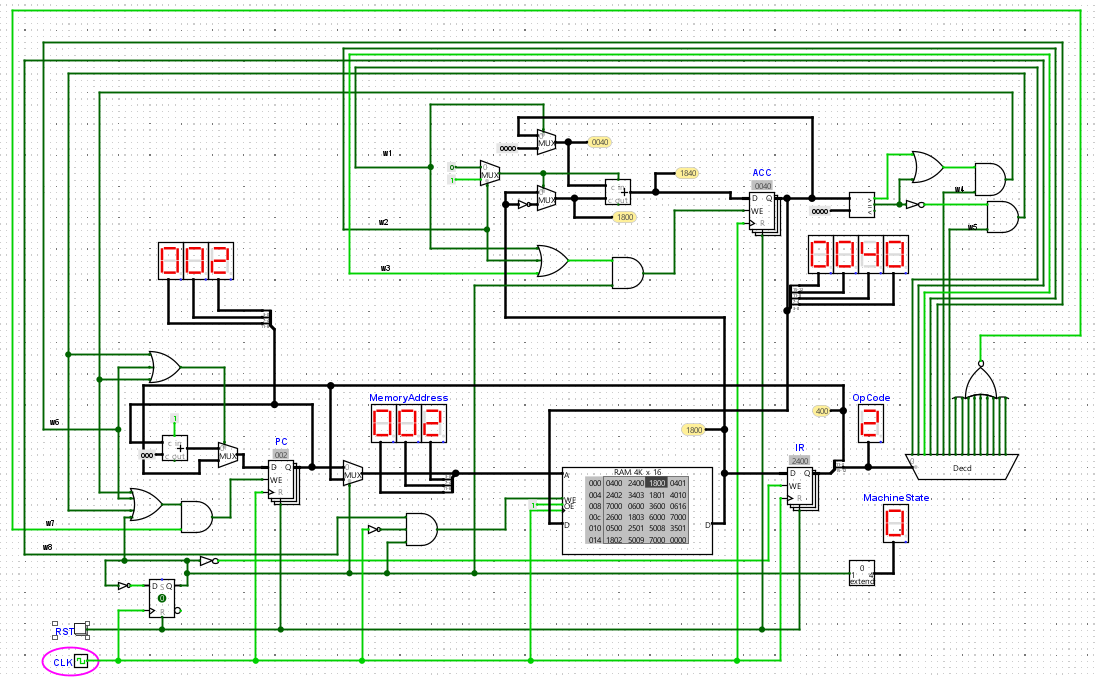
2-1, 7.

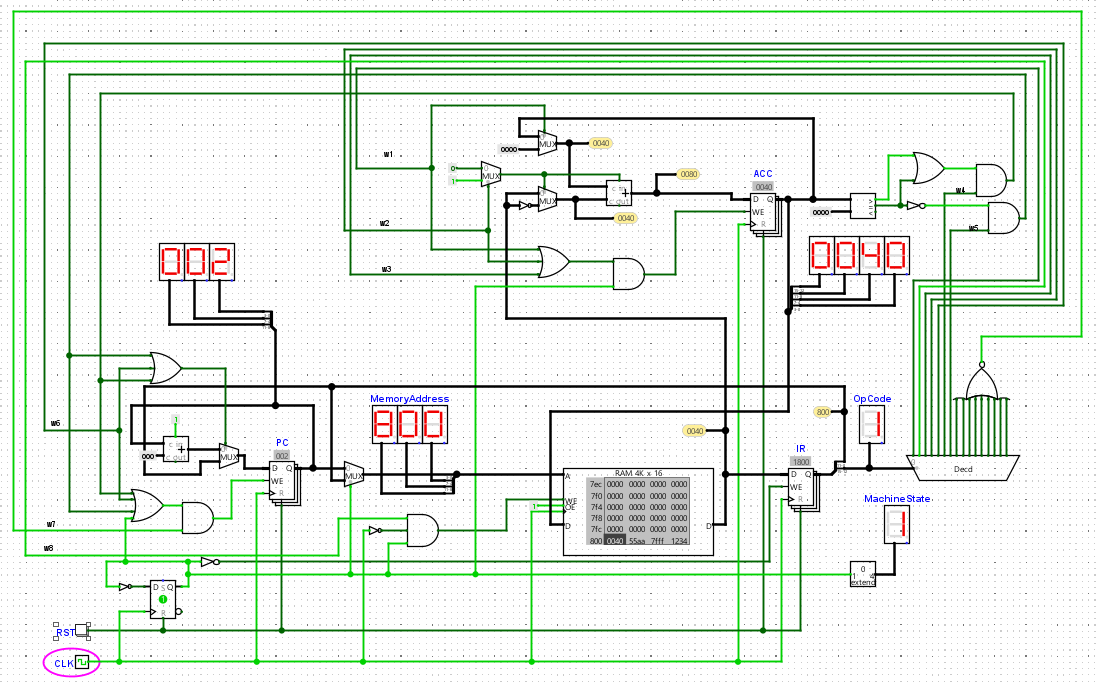
Asynchronous counter가 잘 사용되지 않는 이유는 클럭 펄스가 입력으로 들어가 출력으로 나오는 과정 중에 지연시간이 있기 때문이다. 이를 Propagation Delay라 한다. 이는 비트가 커질수록 심해지는데 각 비트를 차지하는 Flip-Flop을 지날 때 마다 지연시간이 2배로 늘어나는 것을 볼 수 있다.



이렇기 때문에 입력 신호를 받고서 출력 결과를 나타낼 때까지 걸리는 전파 지연 시간이 늘어나 Asynchronous counter가 잘 사용되지 않는다.

**2-1 (MU0)**

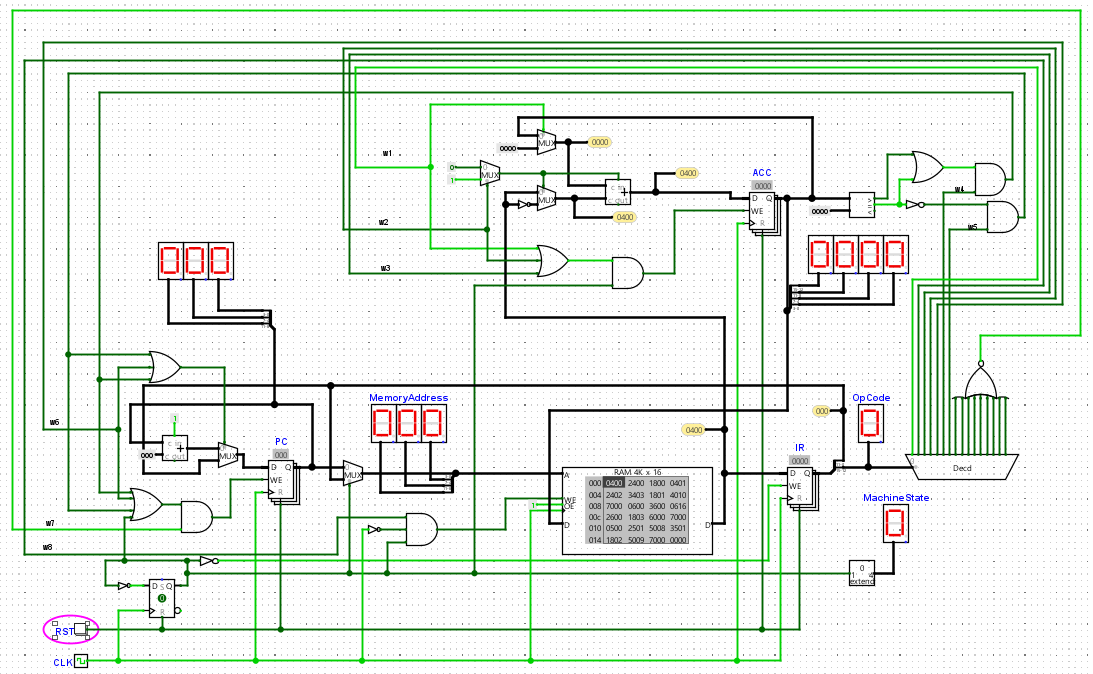


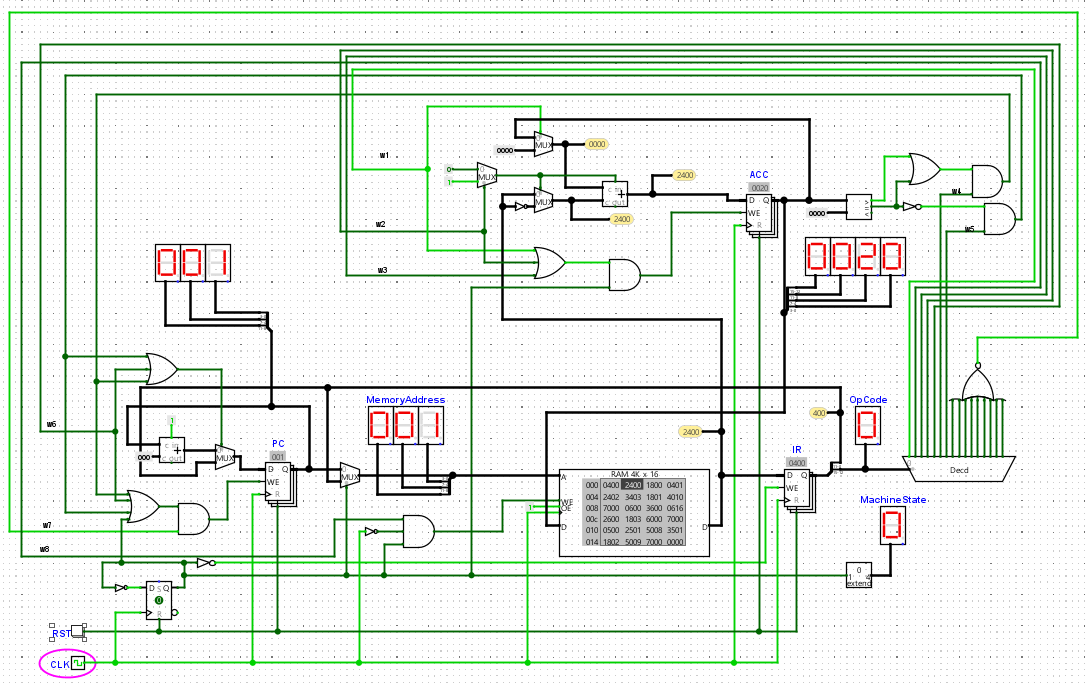


회로의 동작:

위의 그림과 같이 첫 시행에는 Reset을 눌러 초기화를 해준다. 그 후에 CLOCK을 눌러가며 Memory Address, ACC, OpCode의 변화를 살핀다. 여기서 사용된 모든 Flip-Flop은 Rising Edge Flip-Flop이다 그러므로 CLK이 LOW에서 HIGH로 바뀔 때 마다 모든 Flip-Flop의 state가 변한다.

Decoder는 특정 값이 들어갔을 때 하나만 1이고 나머지는 0인 control signal을 출력하는 소자이다. 이를 활용해 끊어진 w1 ~ w8를 연결할 수 있다. w1이 신호 1을 보낸다고 가정할 때, 회로에서 가장 위쪽 adder의 cin에 0이 들어가고 두 인풋 중 하나에는 RAM에서 가져온 값이, 다른 하나에는 0000이 들어가 덧셈을 해 ACC에 넣어주는 것을 확인할 수 있다. 가져온 값에 0000을 더하면 그 값을 그대로 유지하므로 ACC에 RAM에가 가져온 값을 LOAD 했다고 볼 수 있다. LOAD의 OpCode는 0이다. 따라서 Decoder의 Output0은 LDA이다. 이를 그림으로 확인하면 다음과 같다.





다음으로 w2를 확인하자 w2가 1이라 가정하면 위쪽 adder의 cin에 1이 들어가고 두번째 인풋에 메모리에서 가져온 값을 NOT게이트에 통과시켜 가져오는 것을 볼 수 있다. 첫번째 인풋은 ACC의 저장된 값을 그대로 가져온다. NOT게이트에 통과시키고 덧셈 시 1을 추가로 더해주는 것을 보면 양수를 음수로 반드는 2’s complement와 같다. 현재의 ACC값에 메모리에서 가져온 다음 값을 음수를 취해서 더해주는 연산은 뺄셈이다. SUB의 OpCode는 3이므로 Decoder의