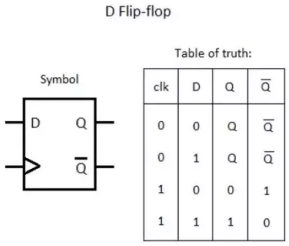
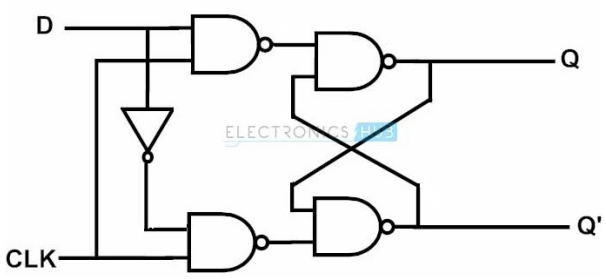
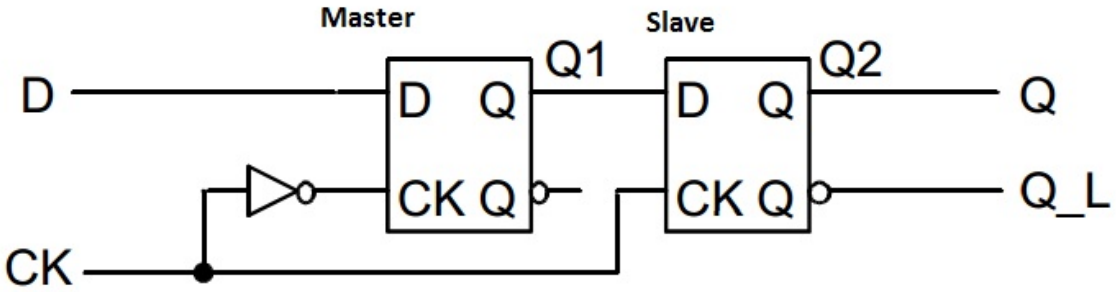
1. D flip flop





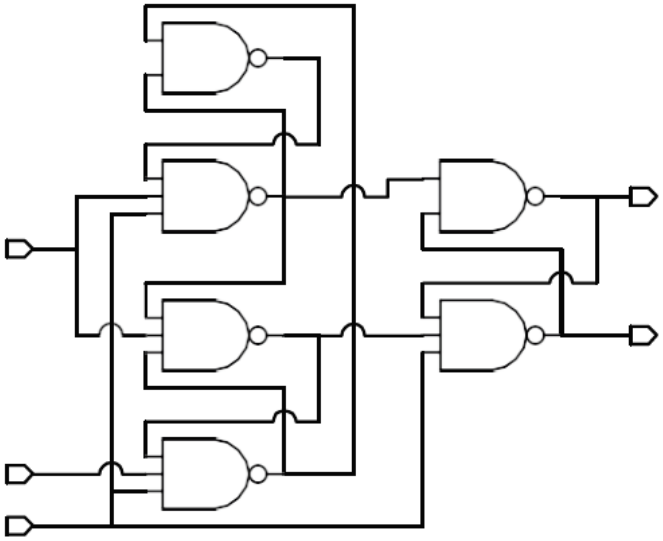
D flip flop은 clk가 0일때는 Q와 Q_bar는 기존의 값을 그대로 가지고 있고, clk가 1이 될 때는 input인 D 값을 그대로 Q로 보내는 (Q_bar는 not D 값으로) 회로입니다.

2. 레지스터 구조

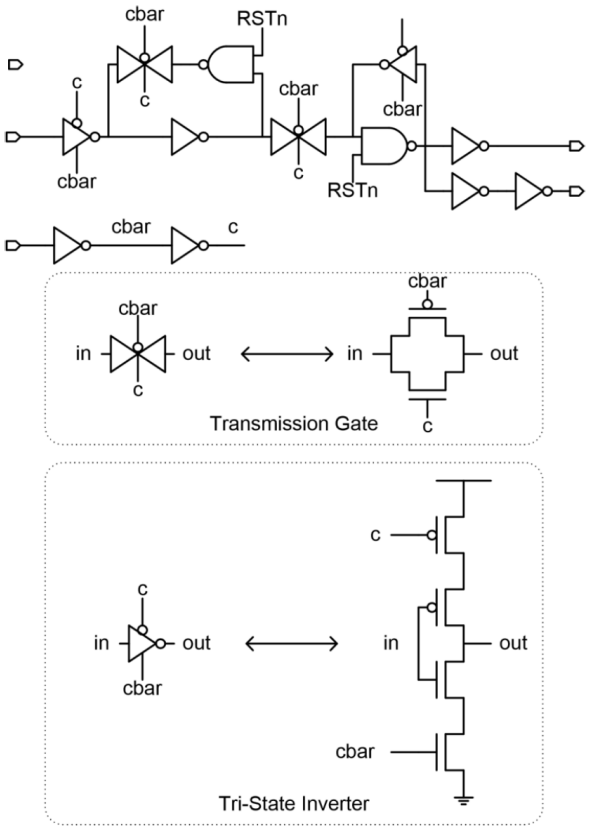


MASTER-SLAVE D FLIP FLOP
i) Positive edge triggered
ii) Note that Slave is controlled by Clk and Master by Clk'
iii) In Negative Edge Triggered, Master is run with Clk

앞에서 말한 D flip flop 두개를 가지고 레지스터를 만들 수 있는데요 이 두개를 같이 연결한 master-slave flip flop이 레지스터의 기본 구조 입니다. 작동원리는 clock이 0일 때는 처음 D flip flop은 Q=D가 되고, 두번째 D flip flop은 꺼져 있기 때문에 Q2 값은 이전 Q2 값을 그대로 가지고 있습니다. 그러다가 clock이 1로 바뀌는 순간 첫번째 flip flop은 clock이 1로 바뀌기 직전의 D=Q1 값을 그대로 가지고 있고, 뒤의 두번째 D flip flop이 켜지게 되면서 Q2는 첫번째 flip flop의 Q1 값을 받아들이게 됩니다. 즉 clock이 0에서 1로 바뀌는 순간의 input인 D 값을 Q에 저장하는 회로입니다.



위의 회로도를 앞에 1번에서 설명드린 D-flip flop 회로도로 구현하면 다음과 같은데요, NAND gate의 경우 속도도 느리고 (logical effort) Area도 많이 차지하기 때문에 이것보다는 아래와 같은 회로도의 flip flop을 많이 씁니다.



3. 레지스터로 SRAM을 못쓰는 이유

SRAM의 경우 A. clock과의 동기화가 어려움 B. Master-slave flip flop은 decoder나 sense amplifier등의 주변 회로들이 필요하지 않고, SRAM은 이런 주변 회로들의 도움이 필요합니다. 그런데 메모리 크기가 작아질수록 메모리 대비 주변 회로들이 차지하는 비율이 높아지기 때문에 매우 비효율적이게 됩니다. 따라서 이런 주변 회로들이 필요 없는 Master-slave flip flop이 레지스터로는 더 효율적입니다.

이거는 조금 오래된거긴 한데 2005년 애플 imac 기준 레지스터와 캐시의 사이즈 및 속도 차이입니다. 최근에는 L1은 512KB까지, L2는 1~4MB사이, L3은 4~20MB정도라고 하고, L3 cache 이후로부터는 eDRAM으로도 구현한다고 하네요.

Read latency: Time to return first byte of a random access

	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency (cycles)	1	3	3	11	160	1E+07
Latency (sec)	0.6n	1.9n	1.9n	6.9n	100n	12.5m
Hz	1.6G	533M	533M	145M	10M	80

4. 레지스터 사이즈

레지스터 사이즈는 마이크로 프로세서의 크기, 명령어 코드(ISA) 의 크기 등에 따라서 달라집니다. 427 수업에서 쓰는 16 bit microprocessor의 예를 들어보면, 일단 데이터 하나하나의 크기가 16bit이기 때문에 한 데이터의 사이즈는 16bit 이겠죠. ISA 또한 데이터로 존재하기 때문에 16bit의 크기를 가집니다. 명령어의 구조는 다음과 같은데요 상위 15-12 bit는 덧셈을 나타내고, 상위 11-8bit와 상위 3-0 bit는 더해지는 숫자들 (A+B 면 A와 B) 이 들어있는 레지스터의 주소를 나타냅니다. 레지스터의 주소가 4 bit이기 때문에 존재하는 레지스터의 숫자는 총 16개가 있겠죠. 이 경우 필요한 총 레지스터의 크기는 16bit (데이터 하나의 크기) X 16 (레지스터 주소의 개수) = 256 bit가 필요하게 됩니다.

				ImmHi/	ImmLo/	
		OP Code	Rdest	OP Code Ext	Rsrc	
Mnemonic	Operands	15-12	11-8	7-4	3-0	Notes (* is Baseline)
ADD	Rsrc, Rdest	0000	Rdest	0101	Rsrc	*