컴퓨터구조실험 보고서

Project #0 - MU0 Processor

과 목 컴퓨터구조실험

담당교수 이성원교수님

학 과 컴퓨터정보공학부

학 번 2021202058

이 름 송채영

제 출 일 2021. 04. 05

1. Introduction

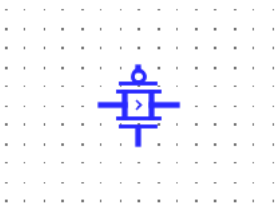
MU0 는 가장 기본적인 processor 구조로, PC, ALU, ACC, IR, Instruction decorder, Control Logic 과 같은 Components 를 포함하고 있다. MU0 는 instruction 을 메모리로부터 가져오는 fetch 와 가져온 instruction 을 해석하고 register 값을 확인하는 decode, decoding 된 instruction 을 실행하는 excute 를 수행한다. MU0 의 instruction set 은 16bit 를 기본으로 하며 4bit 는 opcode 이며 12bit 는 Address 에 해당한다. MU0 의 instruction set 은 다음과 같다.

Instruction	Opcode	Effect
LDA S	0000	ACC := mem[s]
STO S	0001	mem_{16}[s] := ACC
ADD S	0010	ACC := ACC + mem_{16}[s]
SUB S	0011	ACC := ACC - mem_{16}[s]
JMP S	0100	PC := S
JGE S	0101	If ACC >= 0 PC := S
JNE S	0110	If ACC != 0 PC := S
STP	0111	St0-

2. 결과 화면

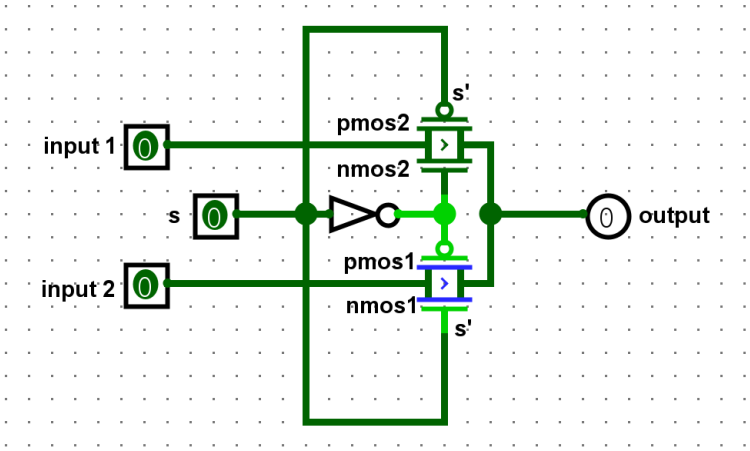
2.1

1. What is the Transmission gate?



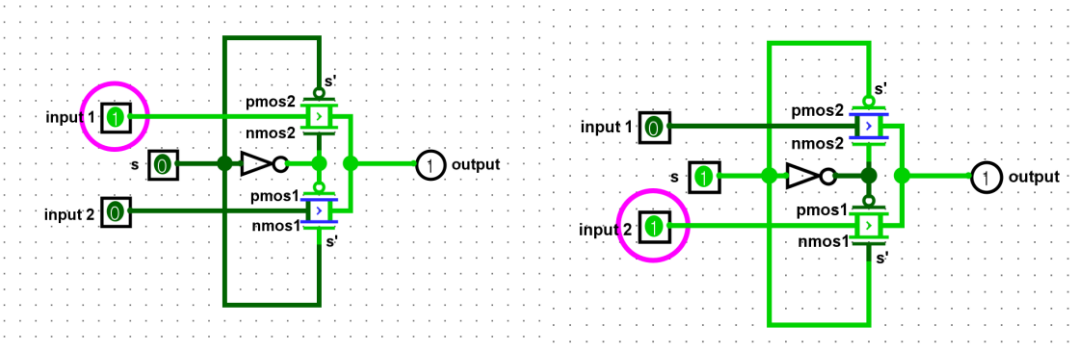
transmission gate 는 pmos 1 개와 nmos 1 개로 이루어진 gate 로 입력을 출력으로 전달해주는 gate 를 말한다. 입력이 0 일 때 nmos 가, 입력이 1 일 때 pmos 가 전달한다.

2. Implement a MUX using Transmission gates



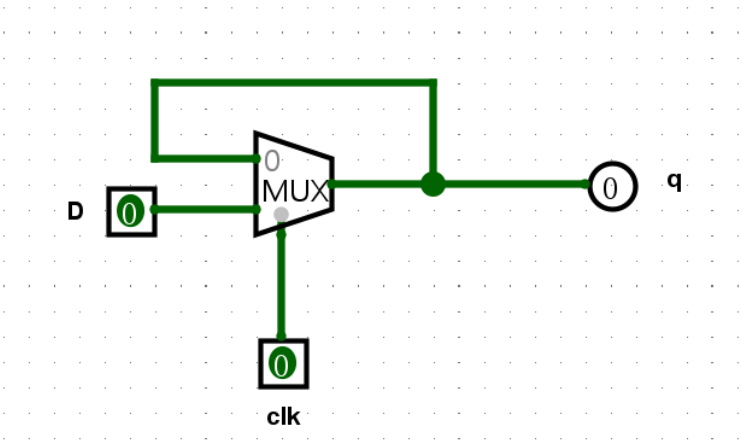
transmission gate 를 활용해서 mux 를 구현하였다. 진리표는 다음과 같다.

S	1	2	Out
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1



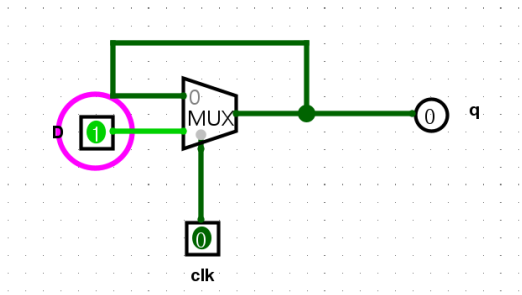
셀렉터가 0 이고 input1 이 1 일 때 input1 의 transmission gate 를 선택하여 1 이 출력되며, 셀렉터가 1 이고 input1 이 1 일 때 input2 의 transmission gate 를 선택하여 1 이 출력된다. 위의 진리표와 동일한 출력 값이 나오는 것을 알 수 있다.

3. Implement a Latch using MUXs

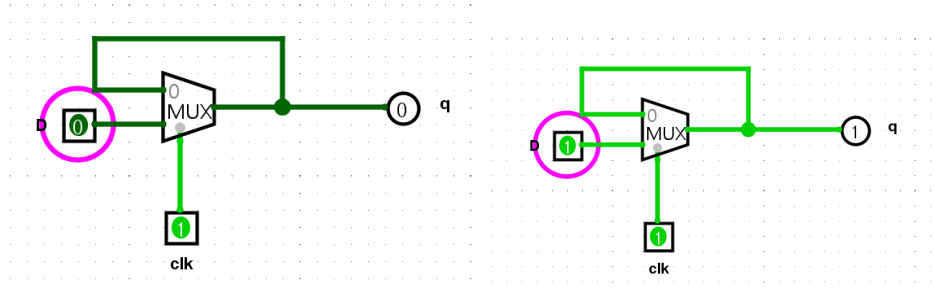


mux 를 사용해서 Latch 를 구현하였다. 입력 데이터 d 와 기존 출력 데이터 q 중 하나를 선택하여 출력하며 진리표는 다음과 같다.

Clk	D	Q
0	X	이전값
1	0	0
1	1	1

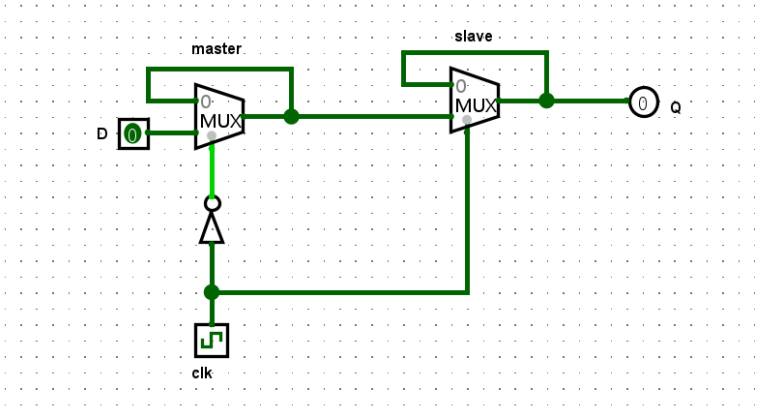


clk 가 0 일 때 이전 출력값인 q 가 출력이 된다.



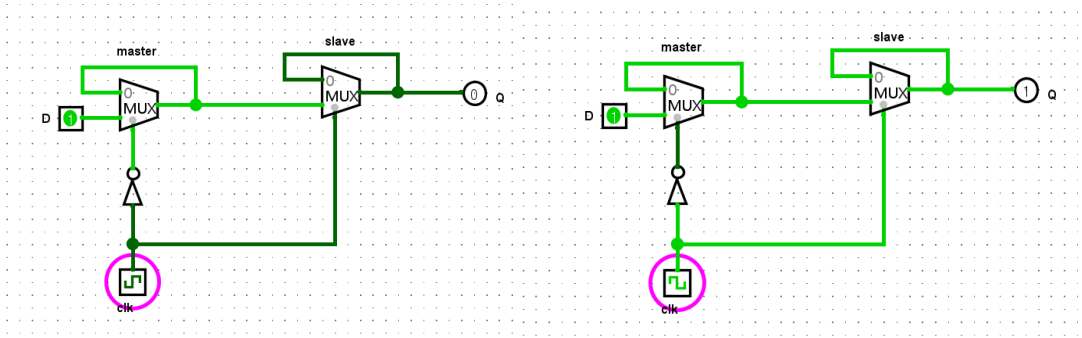
clk 가 1 일 때 D 에 따라 출력이 된다. 즉 D-래치는 멀티플렉서의 결과값을 다시 입력 값으로 주어, CLK 가 1 일 때 결과값이 바뀌고, 0 일 때 이전 값이 출력된다. 위의 진리표와 동일한 출력 값이 나오는 것을 알 수 있다.

4. Implement D-FF using Latches.



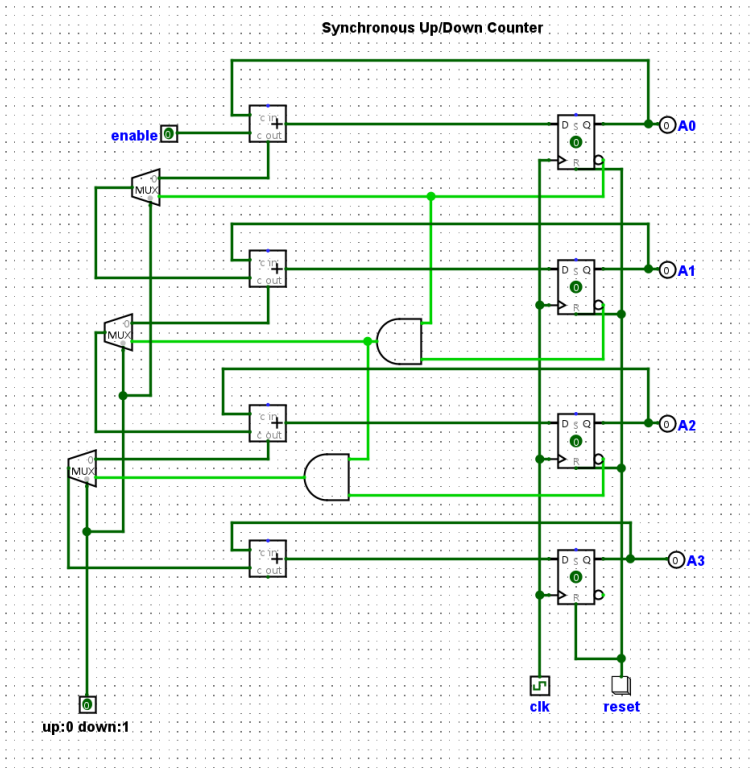
D-Latch 를 사용해서 D filp flop 을 구현하였다. 진리표는 다음과 같다.

Clk	D	Q
↓	0	1
↑	1	1



clk 가 0 일때, 즉 하강엣지일 때 master 가 작동하며 clk 가 1 일 때 즉 상승엣지일때 slave 가 작동한다. 진리표와 동일한 출력 값이 나오는 것을 알 수 있다.

5. Implement Synchronous Up/Down Counter using D-FFs and adders.

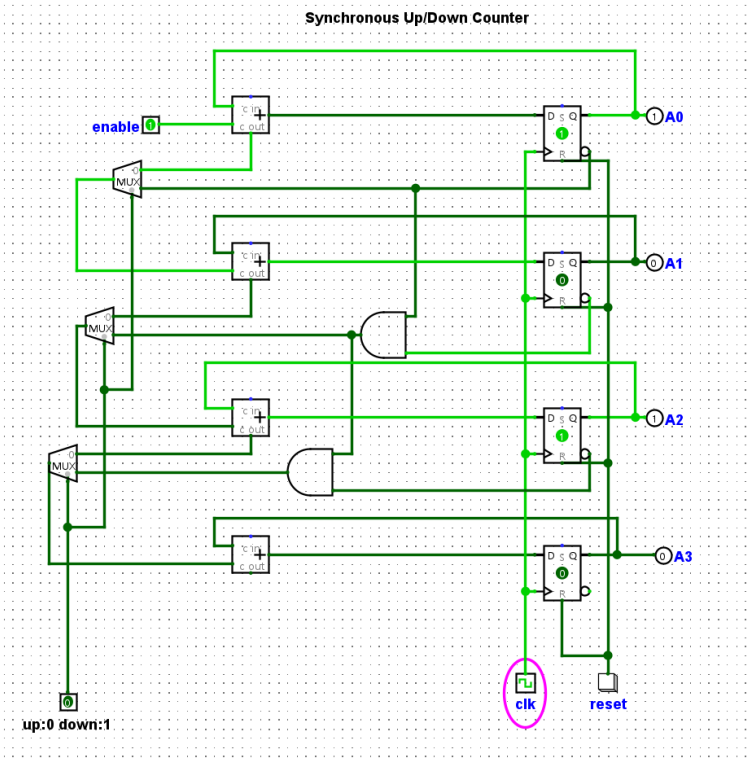


D-FF 와 adder 를 사용해서 Synchronous Up/Down Counter 를 구현하였다. 진리표는 다음과 같다.

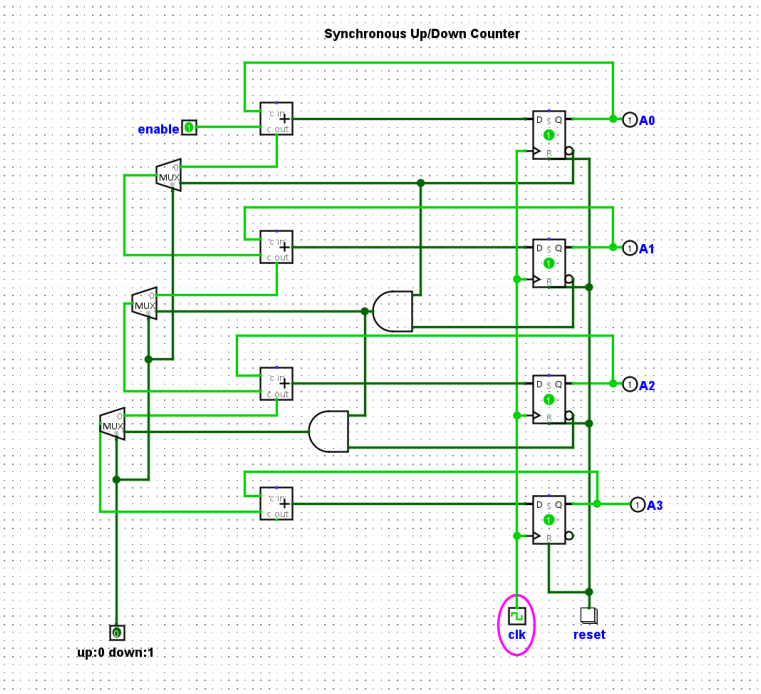
Counter	A3	A2	A1	A0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0

11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

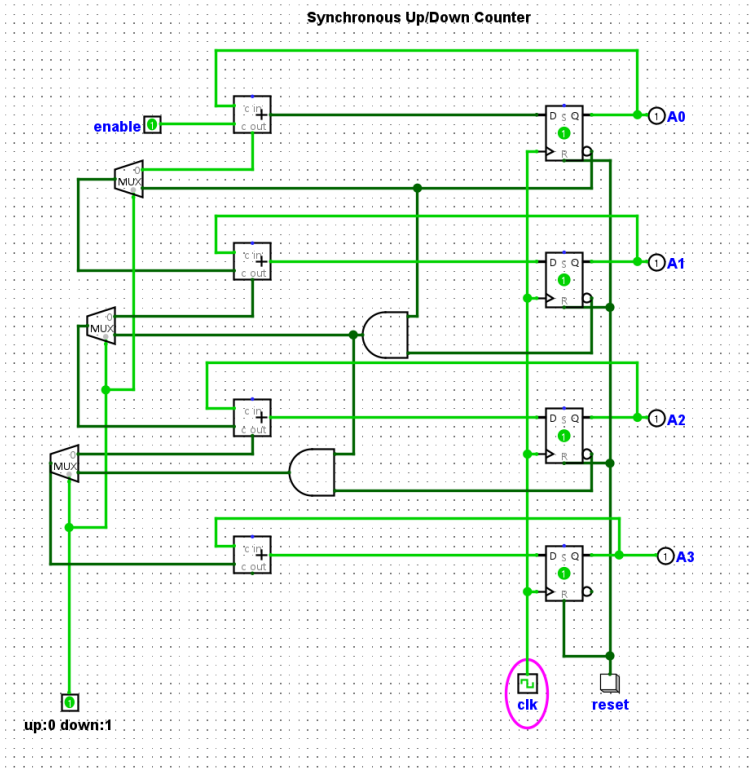
Up 일 때 0 이고 down 일 때 1 인 counter 를 통해서 clk 가 1 일 때 즉 클록이 상승엣지일 때 값을 1 씩 더해준다. 위 진리표는 up 일 때의 진리표 이며 down 일 때의 진리표는 1111 부터 역순으로 count 한다.



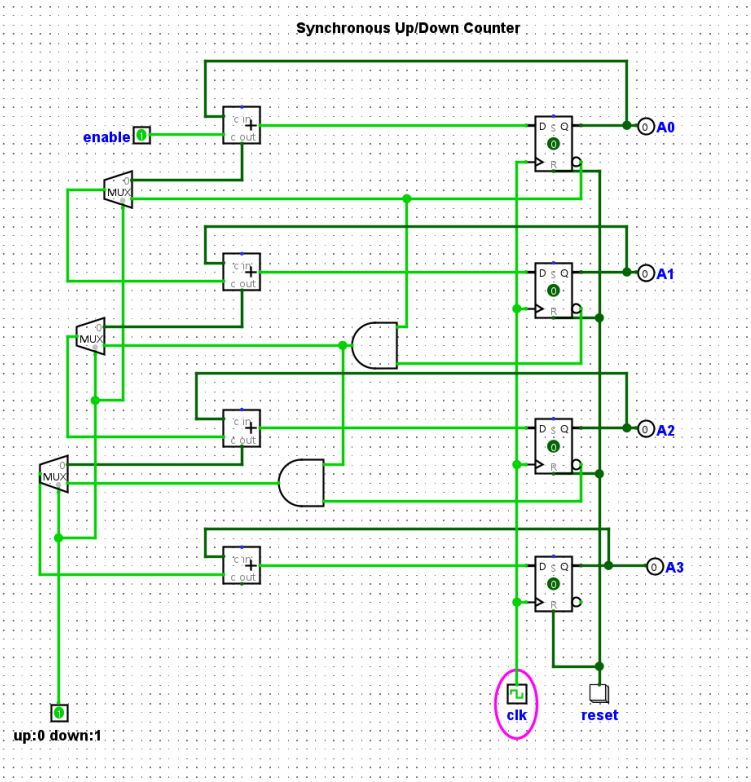
위의 사진은 up 일 때의 결과화면이다.



1111 까지 잘 출력이 되는 것을 볼 수 있다.



위의 사진은 down 일 때의 결과화면이다.



0000 까지 잘 출력이 되는 것을 볼 수 있다.

6. Compare Synchronous counter and Asynchronous counter such as a ripple counter.

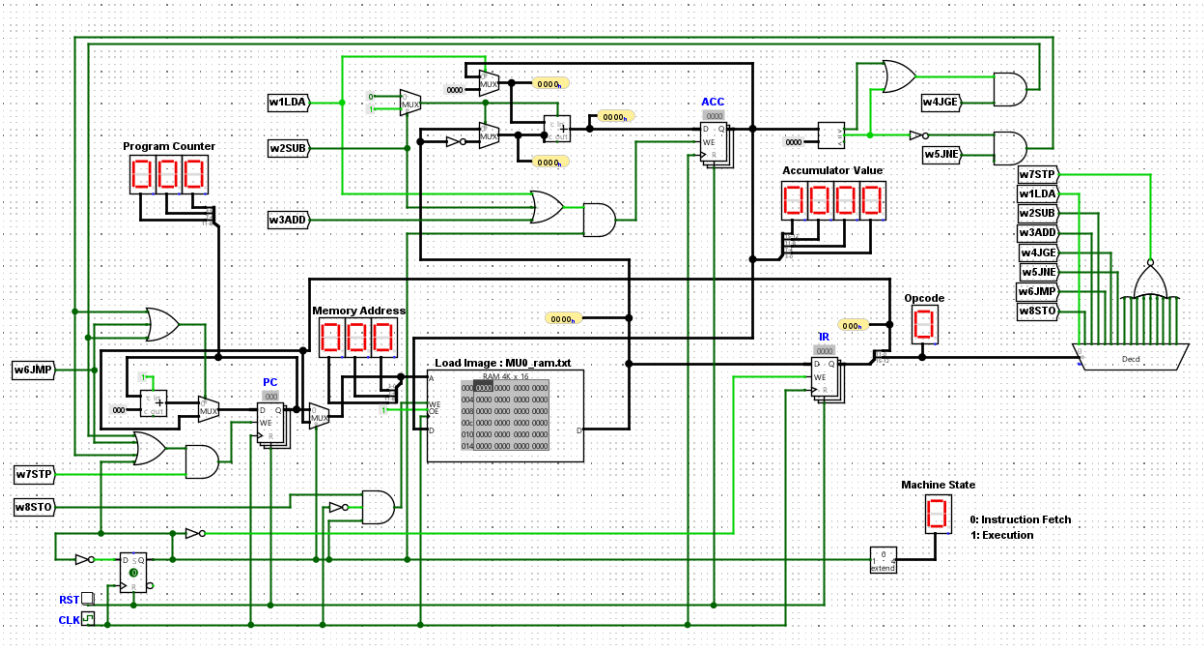
동기식 및 비동기식 카운터는 타이밍과 작동이 다르다. 우선 동기식 카운터는 공통 클럭 신호를 기반으로 작동하는 카운터 유형이다. 카운터의 모든 플립플롭은 동일한 클럭 펄스에 의해 상태가 동시 변경된다. 비동기식 카운터는 각 플립플롭이 이전 플립플롭의 출력에 의해 작동하는 카운터 유형이다. 비동기식 카운터는 카운터 출력이 지연되고 속도가 제한되므로, 동기식 카운터가 비동기식 카운터 보다 빠르고 안정적이다.

7. What makes that Asynchronous counter is rarely used?

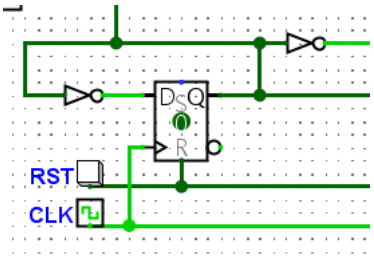
6 번 문제에서도 언급했지만, 비동기식 카운터는 카운터 출력이 지연되고, 속도가 제한된다. 또한 비동기식 카운터는 입출력이 없을 때도 상태를 지속적으로 변경하기 때문에 전력 소모도 동기식 카운터보다 상대적으로 크다. 또한 동기식 카운터보다 회로 설계가 어렵고 시간이 더 많이 소요될 수 있어 비동기식 카운터는 잘 사용되지 않는다.

2.2

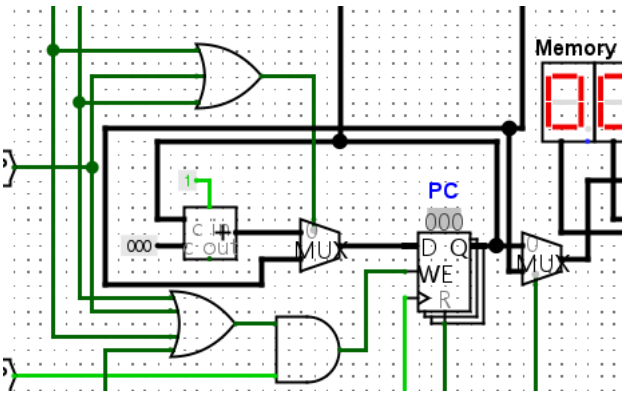
우선 구현한 MU0 의 전체 회로도는 다음과 같다.



MU0 회로의 주요 기능에 대해 설명하면 다음과 같다.

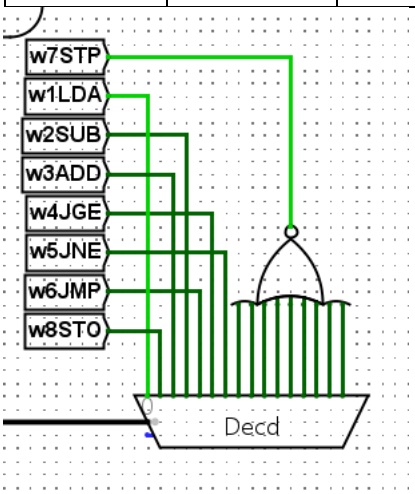


먼저 이 부분은 명령어를 petch 할 때 값을 ACC 나 pc 에 쓰지 않게 하기 위한 D 플립플롭이다. fetch 때 write enable 을 0 으로 execution 때 write enable 을 1 로 해서 올바른 값을 넣어주기 위해 사용한다.

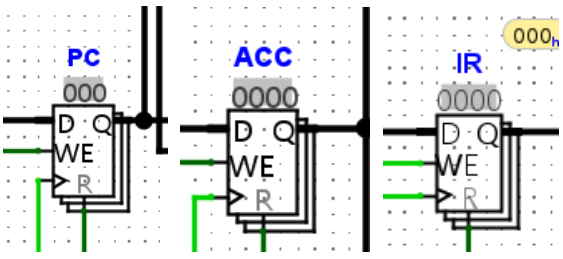


이 부분은 pc 에서 나온 값과 S 를 mux 를 사용해서 선택한다. pc + 1 을 한 값 또는 jump 시 이동할 주소를 받기 전 한 clock 을 벌기 위해서이다. 또한 메모리에 S 에 해당하는 실제 값을 얻어내기 위해서 활용한다. 이를 truth table 로 나타내면 다음과 같다.

a	b	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

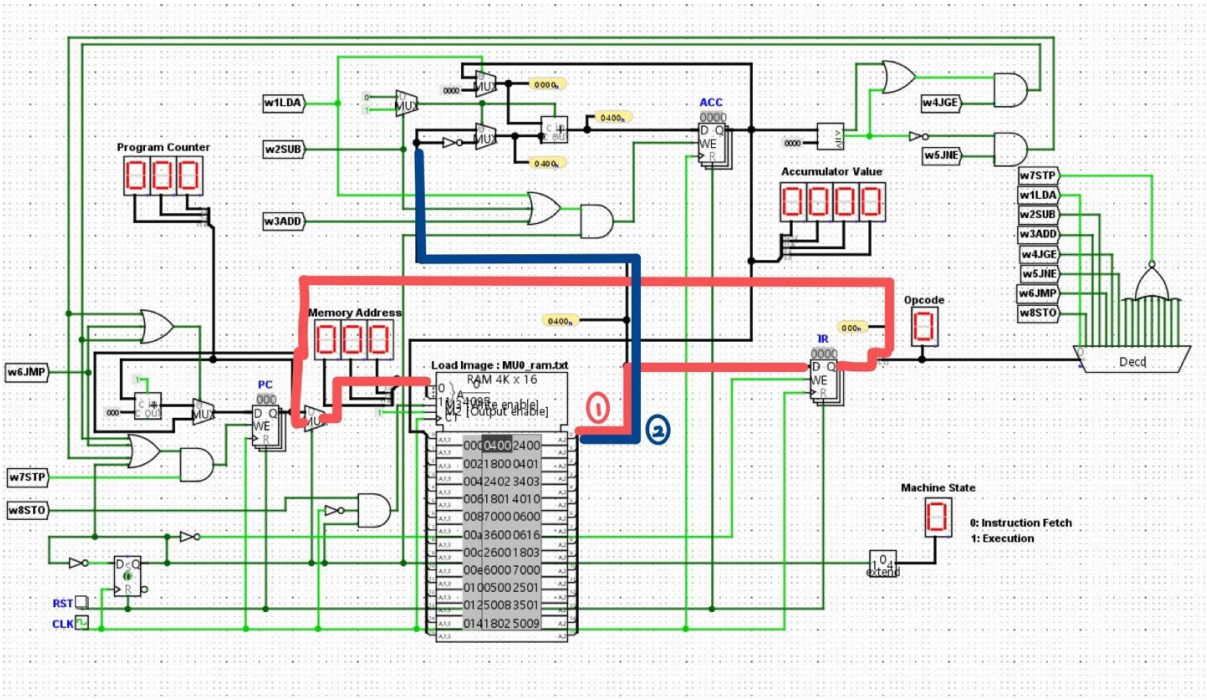


이 부분은 decorder 이다. IR 에 가져온 instruction 을 해석한다. opcode 는 4bit 명령어를 16bit 의 출력으로 바꾸어 작동한다.

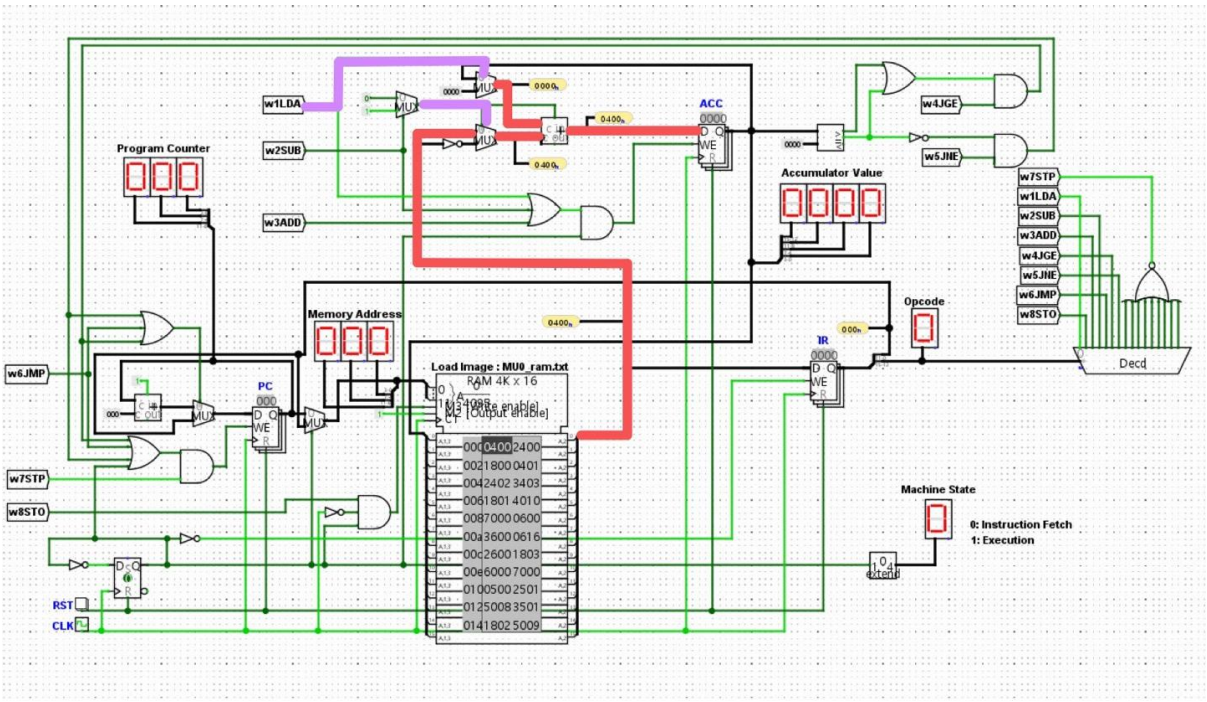


먼저 PC 는 program counter 로 현재 실행하고 있는 instruction 의 주소를 가리킨다. ACC 는 연산에 사용되는 값들을 저장하며, 연산의 결과값을 잠시 저장한다. 마지막으로 IR 은 PC 가 가리키는 instruction 의 주소에서 읽어온 instruction 을 담아두는 기억장소를 말한다. 지금까지 하드웨어의 구성과 기능을 설명해보았다. 이외의 나머지 부분은 명령어를 설명하는 부분에서 추가적으로 설명하겠다.

회로의 동작에 대해 명령어별로 나누어 설명하겠다. 먼저 LDA 명령어이다.

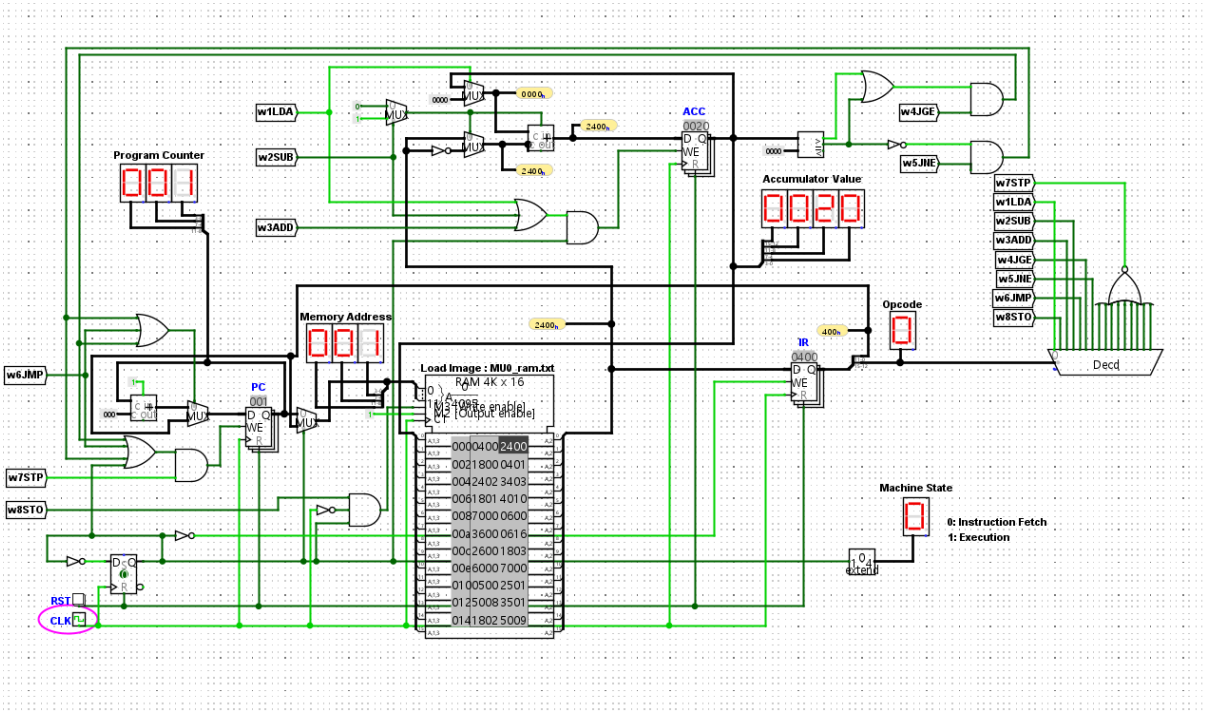


빨간줄인 1 번에 의해 IR 을 지나면 상위 4bit 인 opcode 는 decoder 로 하위 12bit 는 ram 의 Q, 즉 입력으로 간다. 그 후 2 번에는 주소가 저장되어 있는 흐름이다.



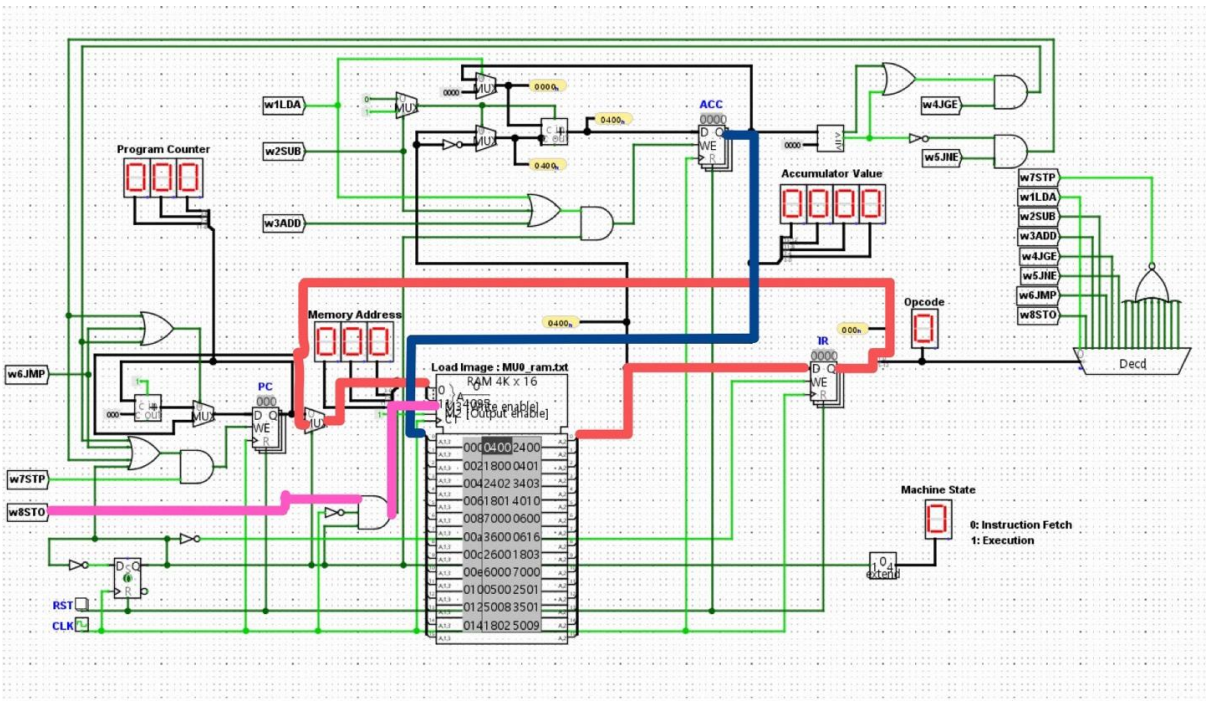
위 사진에서 빨간색 줄은 데이터의 흐름을 나타내었고 보라색 줄은 조건을 나타내었다. 먼저 LDA 가 1 이기 때문에 보라색 선을 따라 0000 을 출력해주고 sub 가 0 이므로 위와

같이 흘러가 더해진다. ACC 에는 주소의 값과 0000 이 더해진 해당 주소의 값이 저장되어 대기한다.

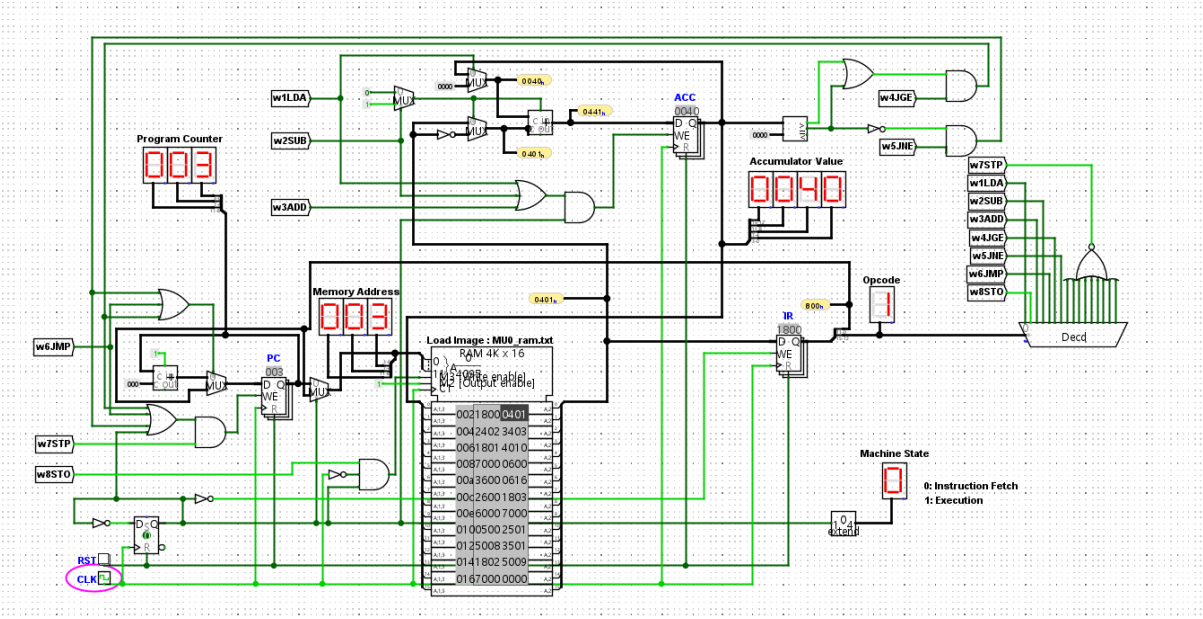


위의 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 0400 instruction 에서 0 은 opcode 로 400 번지 주소에 있는 20 이 ACC 에 대기하고 있는 것을 볼 수 있다.

다음으로 STO 명령어이다.

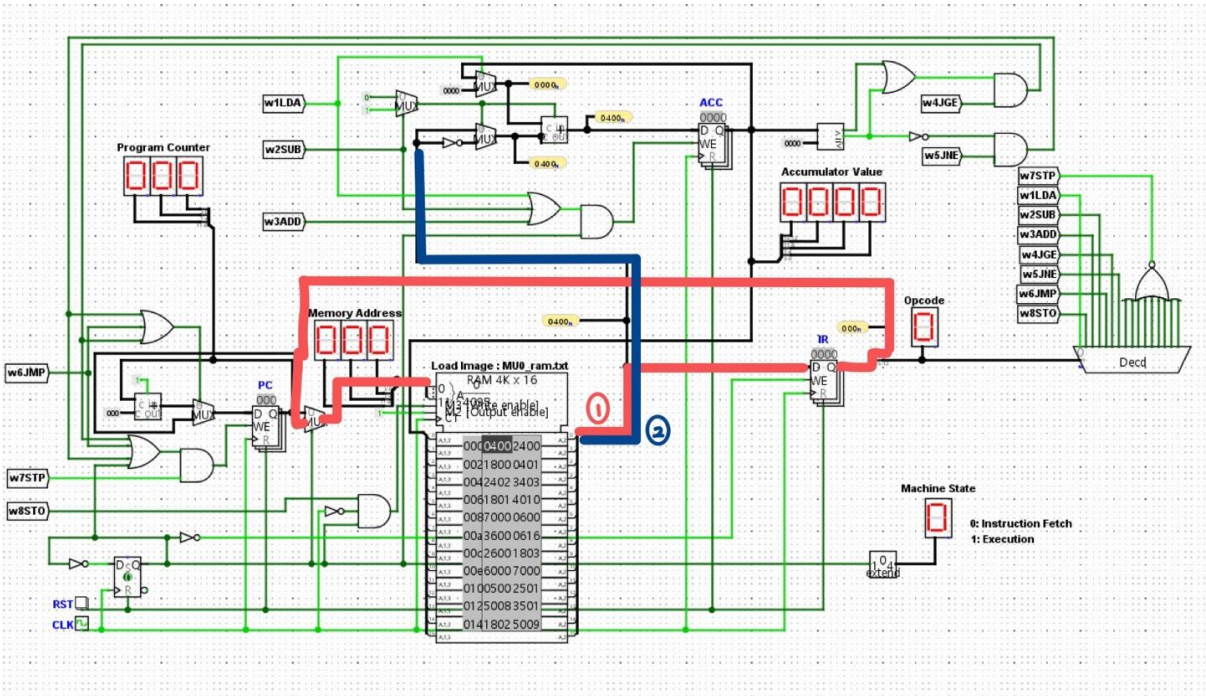


먼저 빨간색 선에 의해 IR 을 지나면 상위 4bit 인 opcode 는 decoder 로 하위 12bit 는 ram 의 Q, 즉 입력으로 값이 저장된 주소 값이 들어간다. 그 후 파란색 선에 의해 ACC 에 대기하고 있고 STO 가 1 이기 때문에 write enable 을 1 로 만들어 저장한다.

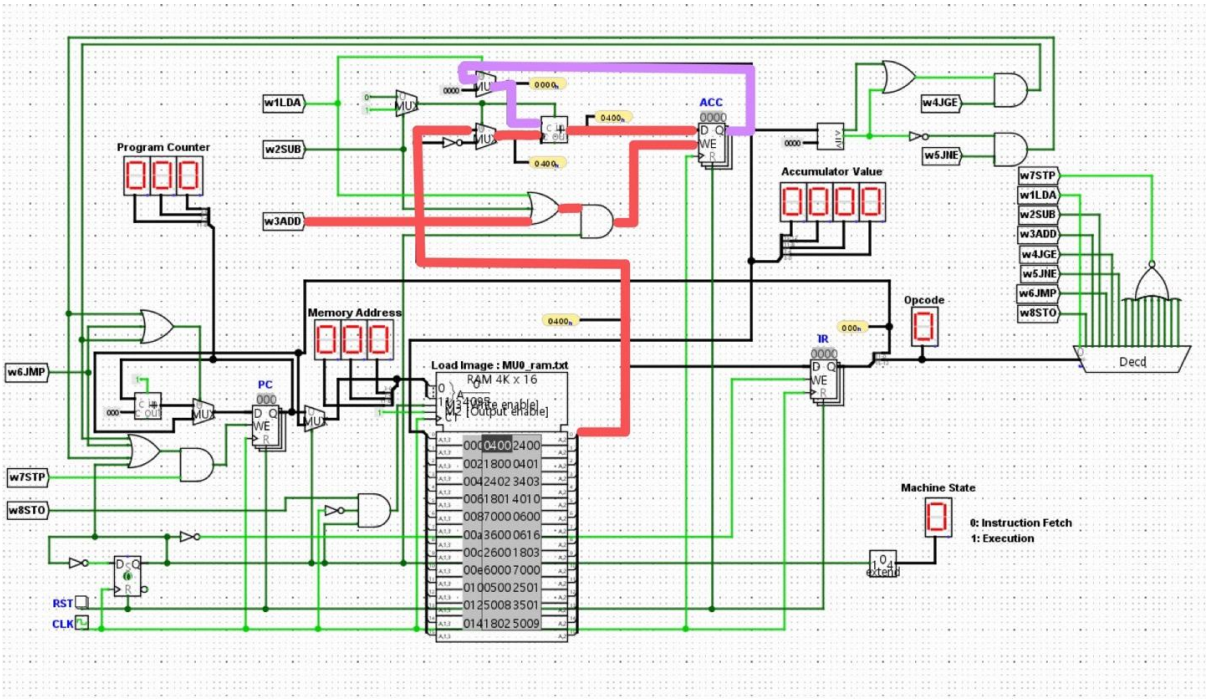


위의 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 1800 instruction 에서 1 은 opcode 로 800 번지 주소에 있는 40 이 ACC 에 대기하고 STO 가 1 일 때 write enable 이 1 이 되어 저장되는 것을 볼 수 있다.

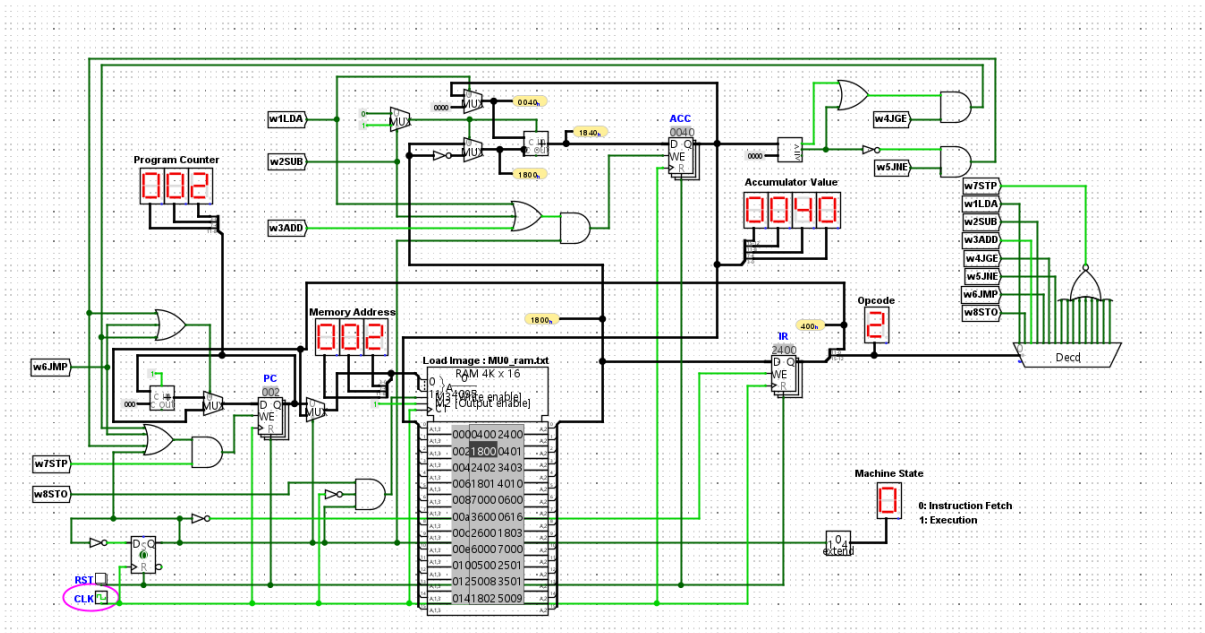
다음으로 ADD 명령어이다.



빨간줄인 1 번에 의해 IR 을 지나면 상위 4bit 인 opcode 는 decoder 로 하위 12bit 는 ram 의 Q, 즉 입력으로 간다. 그 후 2 번에는 주소가 저장되어 있는 흐름이다.

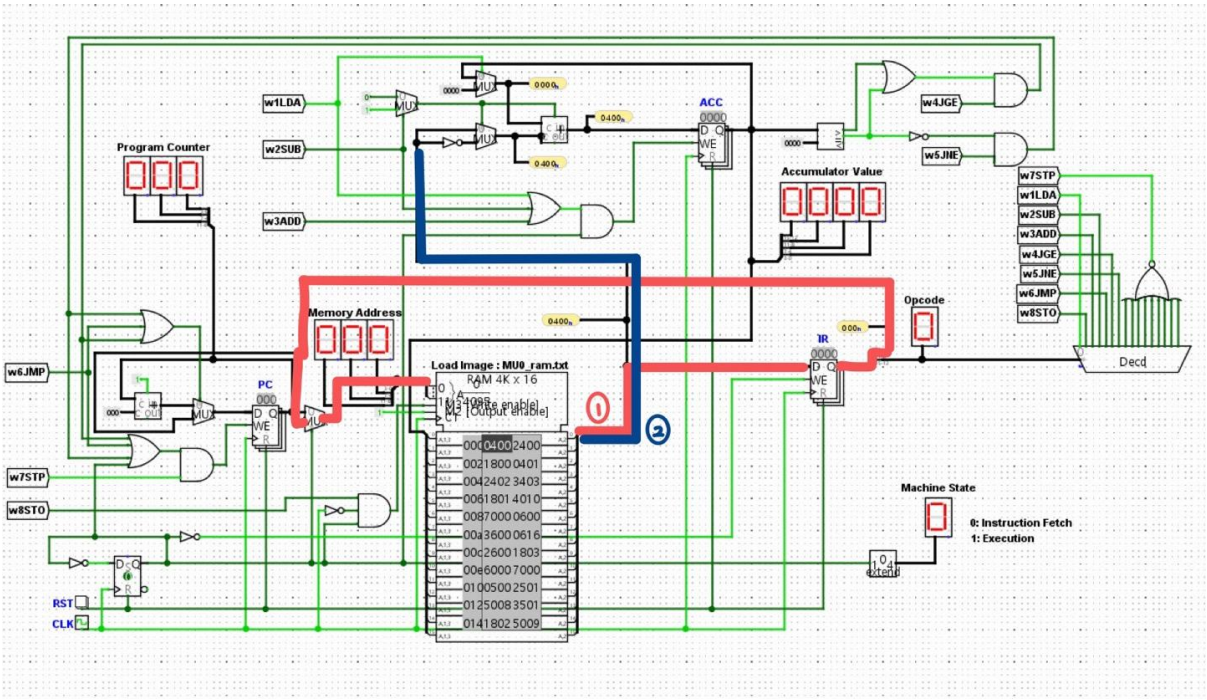


ACC 에 저장되어 있는 값이 보라색 선을 따라 흐른다. 이때 LDA 는 0 이다. 빨간색 선을 보면 SUB 가 0 이므로 adder 에서 보라색 선과 빨간색 선이 만나 더해진 후 ACC 에 대기한다. ADD 가 1 이므로 더해진 값을 write enable 할 수 있다.

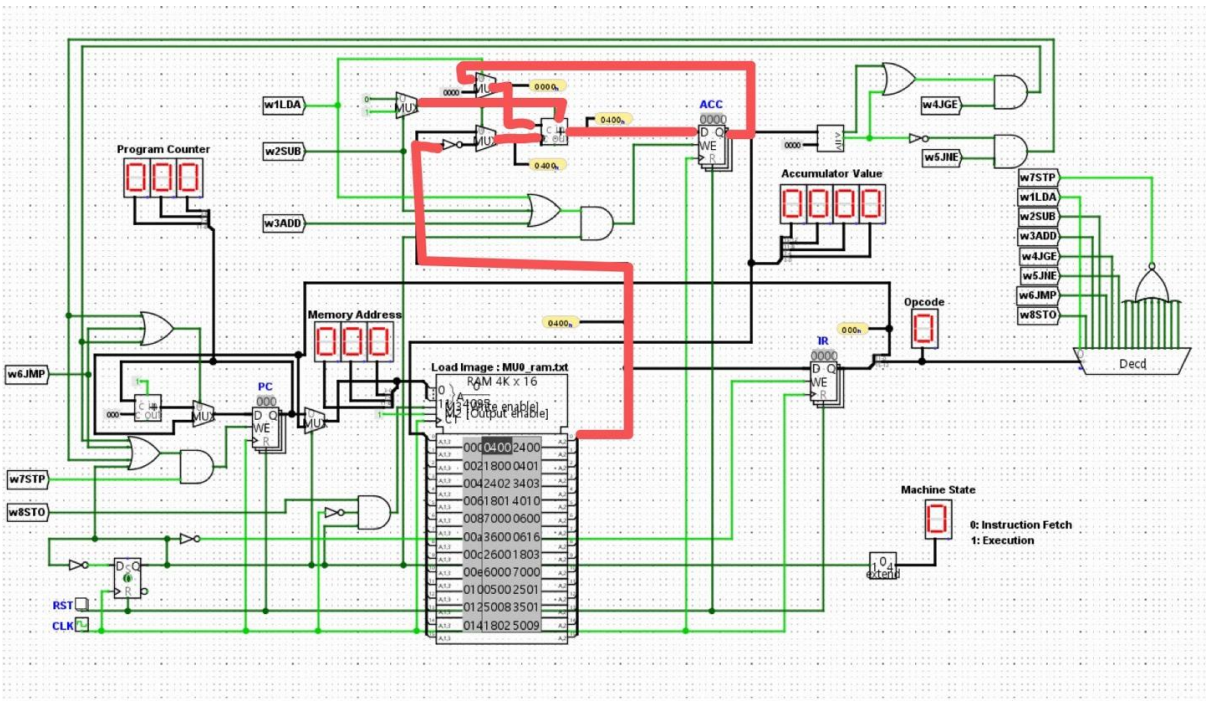


위의 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 2400 instruction 에서 2 는 opcode 로 400 번지 주소에 있는 20 과 전에 저장되어 있던 20 이 ADD 되어 ACC 에 40 이 되므로 결과가 잘 나온 것을 알 수 있다.

다음으로 sub 명령어이다.

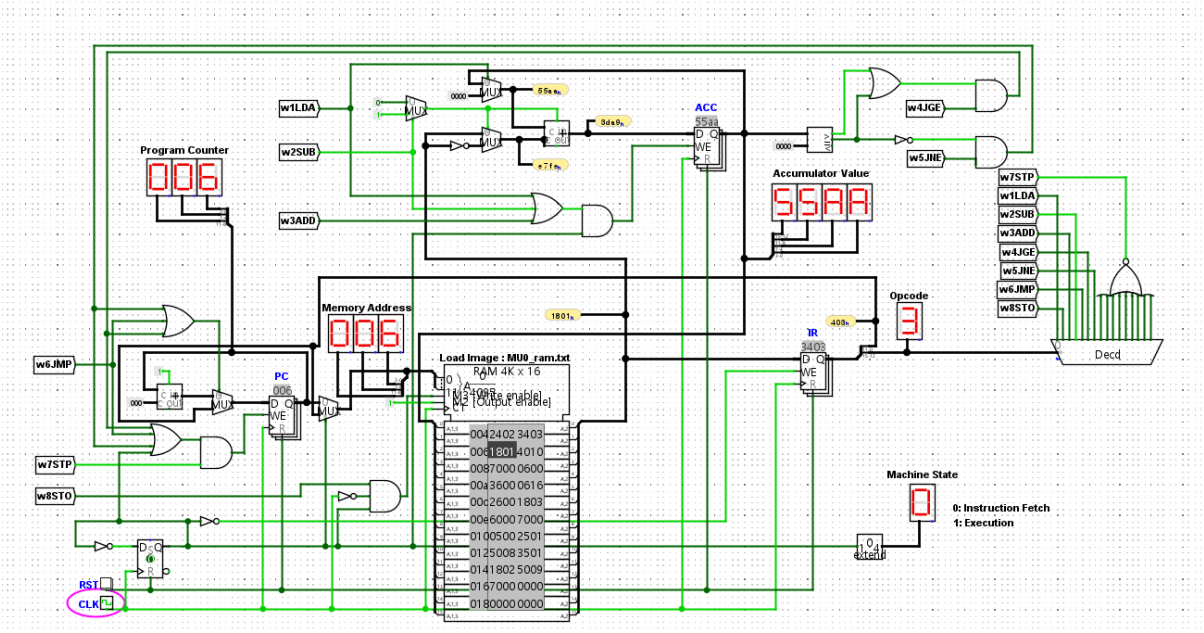


빨간줄인 1 번에 의해 IR 을 지나면 상위 4bit 인 opcode 는 decoder 로 하위 12bit 는 ram 의 Q, 즉 입력으로 간다. 그 후 2 번에는 주소가 저장되어 있는 흐름이다.



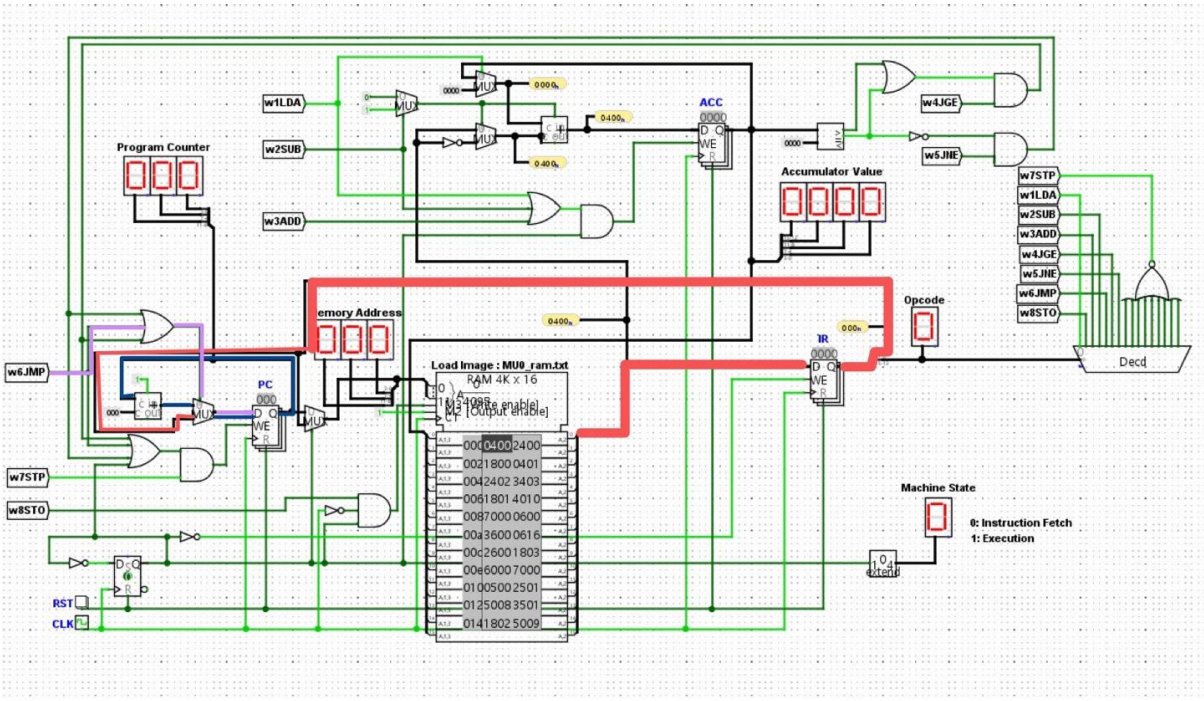
ADD 와 동작 방식이 비슷하다. ACC 에 저장되어 있는 값이 mux 로 들어가고 sub 가 1 이므로 not gate 를 지나서 mux 를 통해 나온다. 이때 adder 에 sub 가 1 이므로

constant 1 값이 들어온다. 이것은 2's complement 처럼 반전 후 1 을 더해주는 역할이다. 뺄셈을 진행해준 후 ACC 에 대기한다.

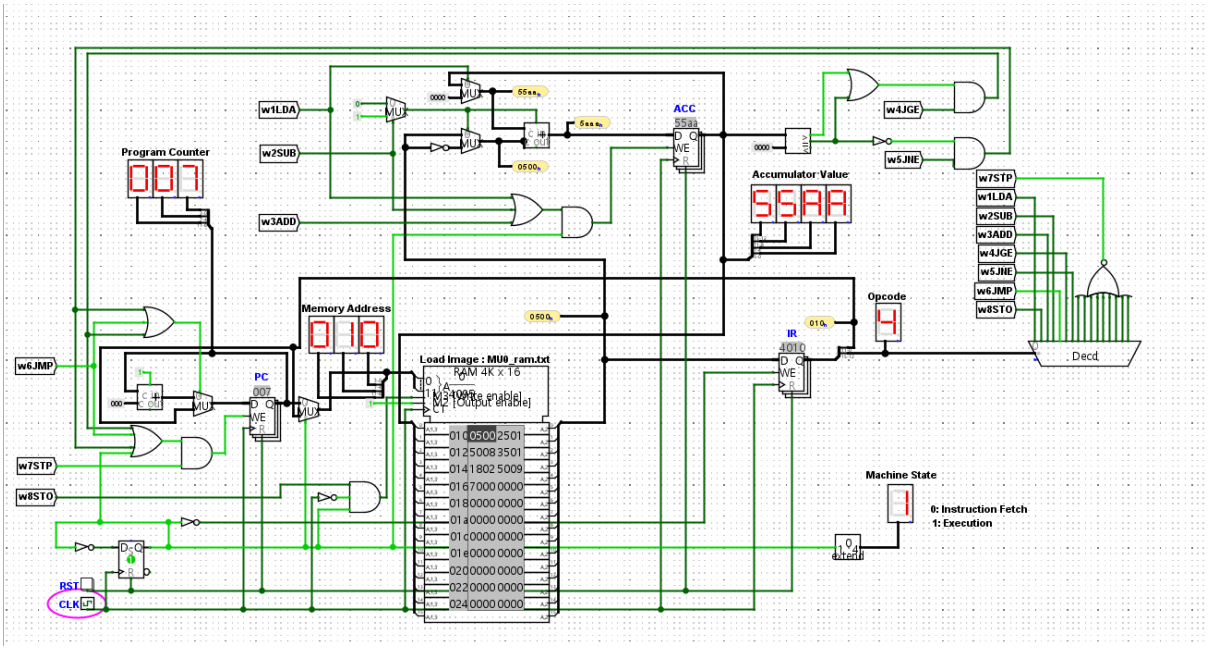


위의 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 3403 instruction 에서 3 은 opcode 로 403 번지 주소에 0055 가 있다. 전에 저장되어 있던 55ff 에서 0055 를 빼고 1 을 더하면 55AA 가 되므로 sub 가 잘 이루어졌다.

다음으로 jmp 명령어이다.

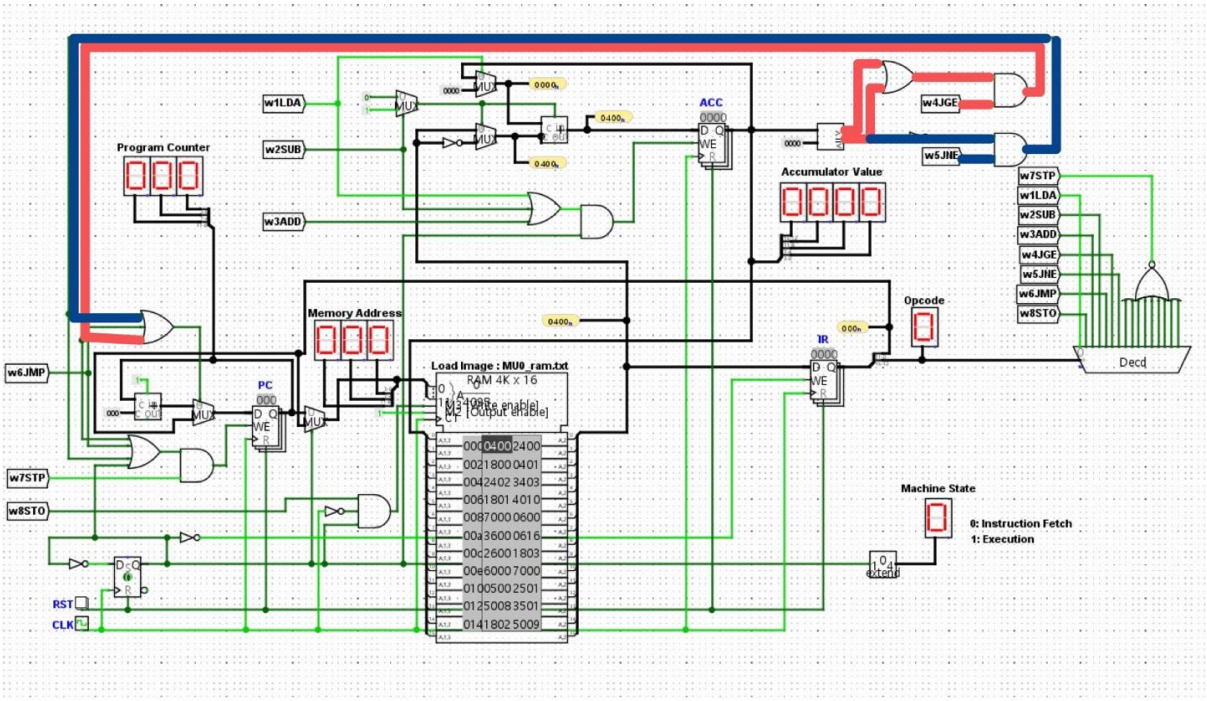


빨간색 선은 해당 주소의 값이며 파란색 선은 pc 보라색 선은 jump 가 1 인지 0 인지를 알려준다. 파란색일 경우 jmp 가 아니므로 pc + 1 을 해주며 보라색일 경우 해당 주소로 jump 한다.

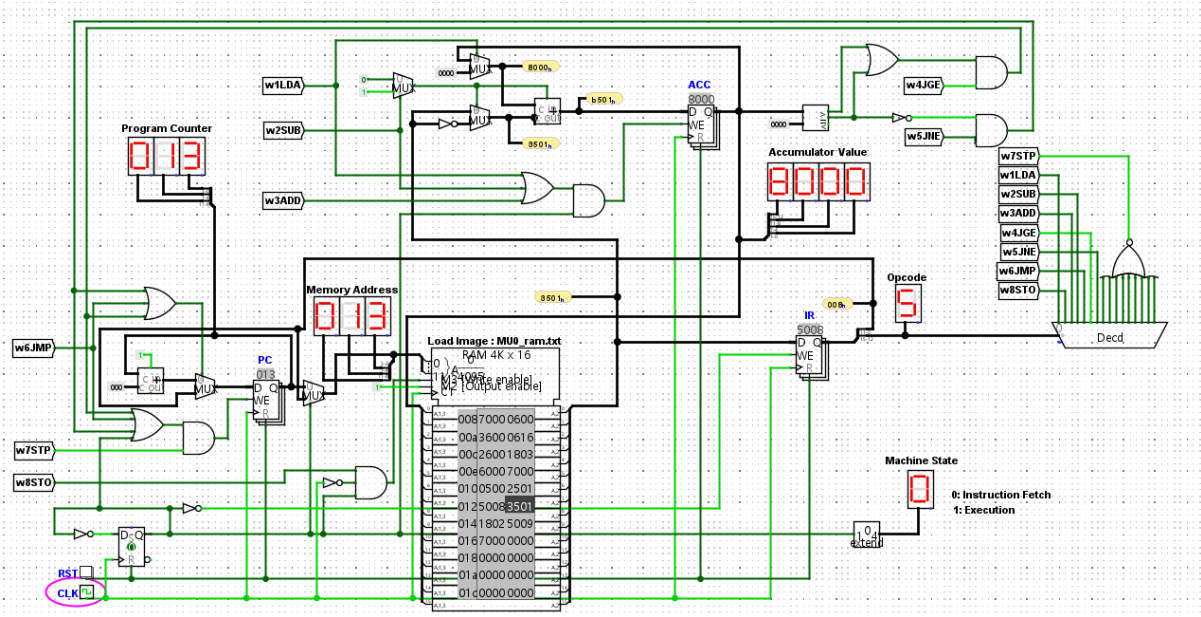


위의 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 4010 instruction 에서 4 는 opcode 로 010 번지 주소로 jump 하면 0500 이므로 다음에 0500 instruction 을 수행해야 한다.

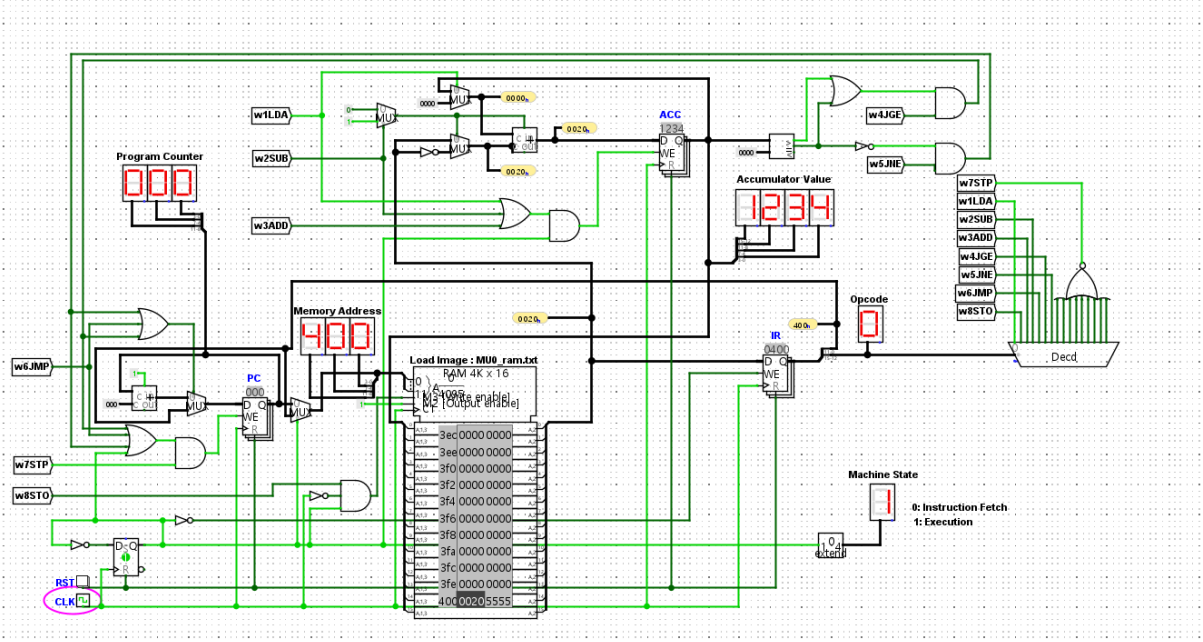
다음은 JGE 명령어와 JNE 명령어이다.



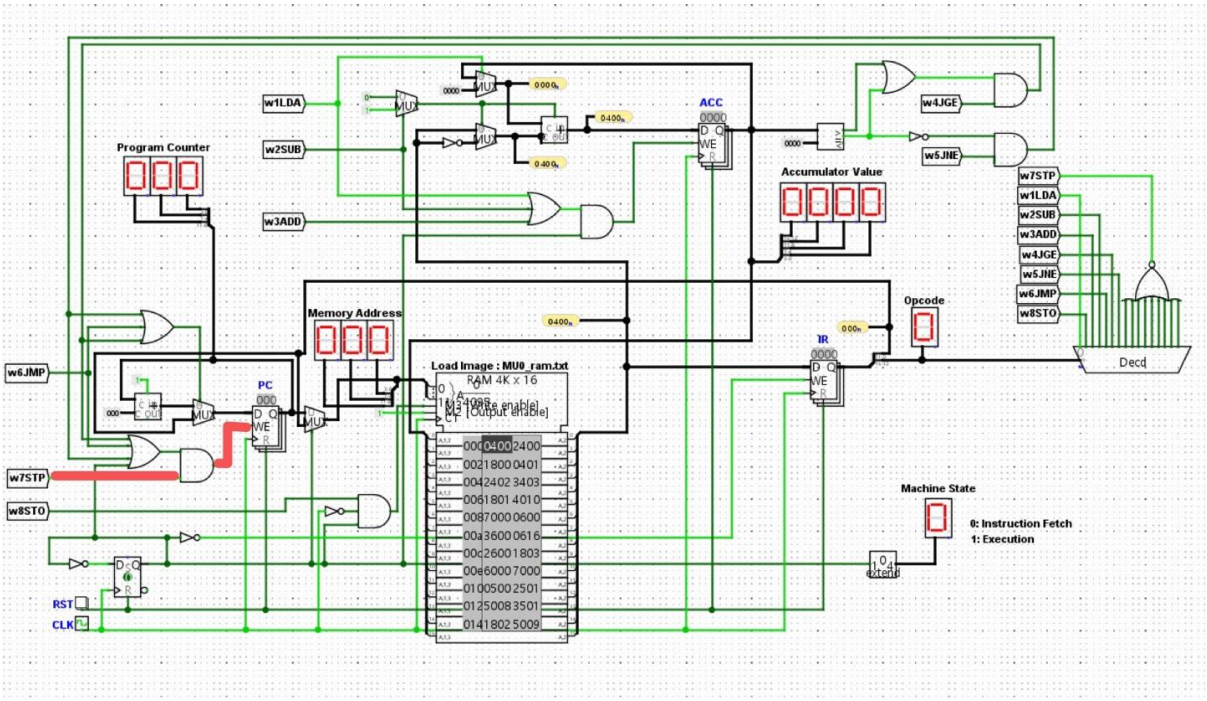
빨간색선은 JGE 명령어이며, 파란색 선은 JNE 명령어이다. JGE 는 ACC 의 값이 0 보다 크거나 같을 때 jump 하며 JNE 는 0 이 아닐 때 jump 한다. JGE 는 comparator 을 통해 ACC 의 값을 확인한다.



JGE 일 경우 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 5008 instruction 에서 5 는 opcode 로 간다. 이때 ACC 의 값이 8000 이므로 0 보다 작으므로 jump 하지 않고 013 번지로 넘어간다.



JNE 일 경우 회로의 주요 기능에서 설명했던 것처럼 pc + 1 이 되었고, 6000 instruction 에서 6 은 opcode 로 간다. ACC 의 이전 값이 1234 이므로 다음에 0400 으로 jump 한 후 다음에 0400instruction 을 수행해야 한다.



마지막으로 STP 명령어이다. opcode 가 7 일 때 pc 의 write enable 이 항상 0 이여서 명령어 실행을 멈춘다.

RAM 에 load 된 txt 사용하여 명령어를 MU0 어셈블리어로 작성하고 작동 방식을 설명한 것이다.

주소	명령어	opcode	주소의 실제 값	ACC	실행순서
0000	0400	LDA	400	20	1
0001	2400	ADD	400	40	2
0002	1800	STO	800	40	3
0003	0401	LDA	401	5555	4
0004	2402	ADD	402	55FF	5
0005	3403	SUB	403	55AA	6
0006	1801	STO	801	55AA	7
0007	4010	JMP	10	55AA	8
0008	7000	STP	0	x	x
0009	0600	LDA	600	1234	15
000A	3600	SUB	600	0	16
000B	0616	LDA	616	0	17
000C	2600	ADD	600	1234	18
000D	1803	STO	803	1234	19
000E	6000	JNE	0	1234	20
000F	7000	STP	0	x	x
0010	0500	LDA	500	7FFF	9

0011	2501	ADD	501	8000	10
0012	5008	JGE	8	8000	11
0013	3501	SUB	501	7FFF	12
0014	1802	STO	802	7FFF	13
0015	5009	JGE	9	7FFF	14
0016	7000	STP	0	x	x

- 제안서에서 요구한 Memory 설계 바꾸는 방법

rom 은 0 부터 7ff 까지의 주소이고, ram 은 800 부터 fff 까지의 주소이므로, 저장장치로 들어가는 12bit 를 dmux 로 분할하여 연결한 후 12bit 중 상위 1bit 를 selecter 로 연결하여 0 일 때 rom 으로, 1 일 때 ram 으로 가게 수정하여 구현하고 각각의 저장장치에 출력값 또한 mux 로 연결하여 dmux 와 동일한 selecter 를 가지면 프로젝트 제안서에서 원하는 메모리의 설계를 할 수 있다. 또한 ram 에 clear pin 을 활성화하여 reset 버튼과 연결하면 휘발성 메모리의 특성 또한 반영할 수 있다.

3. 고찰

이번 프로젝트를 진행하면서 Logisim 의 다양한 design 에 대해 알 수 있었다. 제안서에 주어진 회로만 봤을 때 이게 어디에 위치한 것이고 무엇을 사용해야 하는지 감이 잘 오지 않았는데, 하나하나 그려보고 data bit 를 바꾸고 연결해가며 회로에 대해 더 이해할 수 있었던 것 같다. 선을 하나하나 따라 가보며 이게 왜 이렇게 작동하는지 이해할 수 있었고 무엇보다 MU0 의 작동방식에 대해 더 자세하게 알게 된 것 같다. 회로를 짜면서 Tunnel 의 이름이 동일하지 않아 회로가 완성이 되지 않았다. 어디서 문제가 생기는지 확인하기 위해 많은 시간을 썼지만 생각보다 단순하고 당연한 곳에서 문제가 생겨 한편으로는 허무했지만, 다시는 이런 사소한 실수를 안 할 것 같다.

4. Reference

컴퓨터구조실험 강의자료