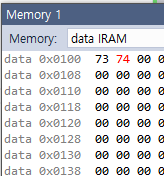
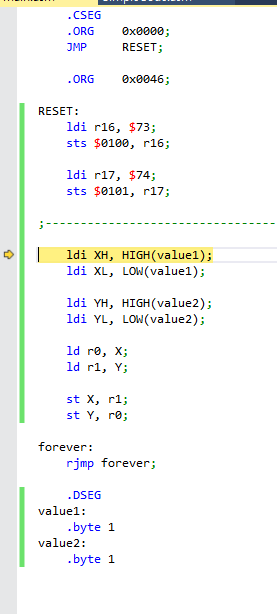
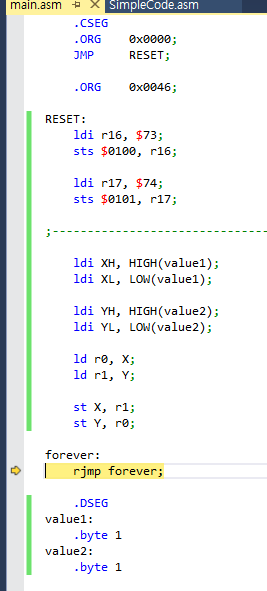
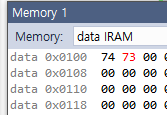
Lab 3-1



Input 값 대입

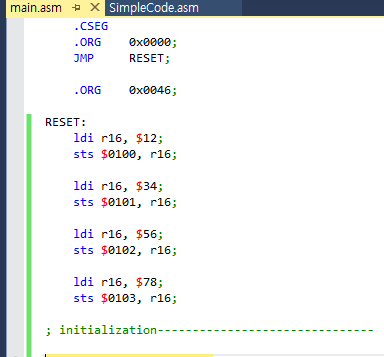
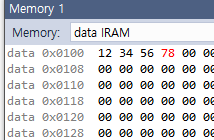
value 들을 swap 시켜주었다.

X레지스터의 0x0100번지의 주소를 할당하고 Y레지스터의 0x0101번지의 주소를 할당한다.

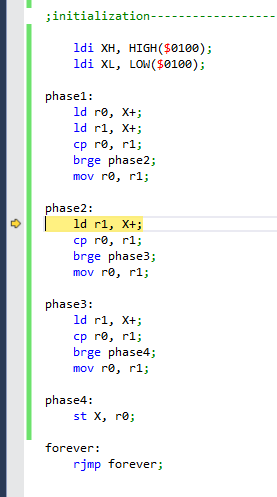
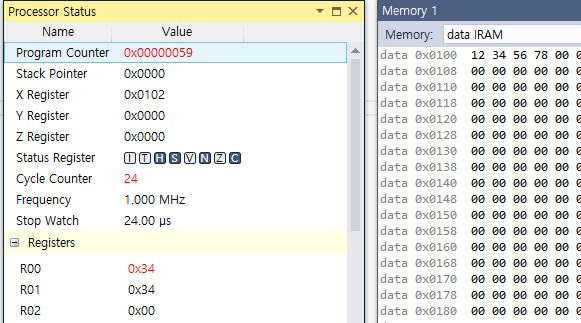
그 뒤 X주소에 해당하는 정보를 r0에 저장하고, Y주소에 해당하는 정보를 r1에 저장,

그 후 X주소에 r1을 저장하고, Y주소에 r0을 저장하는 방식으로 swap 하였다.

Lab 3-2

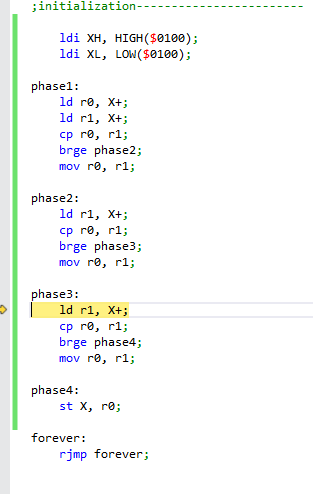
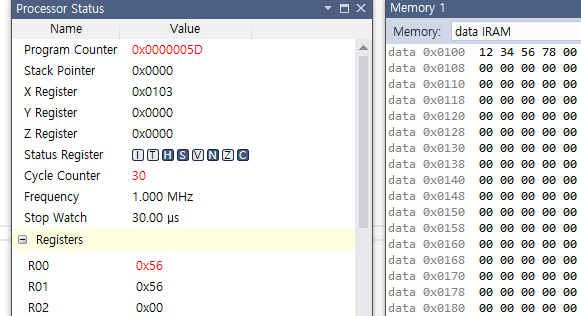
 

주어진 조건으로 초기화를 진행

phase1 루프를 수행하고 난 뒤 (r0이 r1보다 작으므로 mov 명령어 수행)

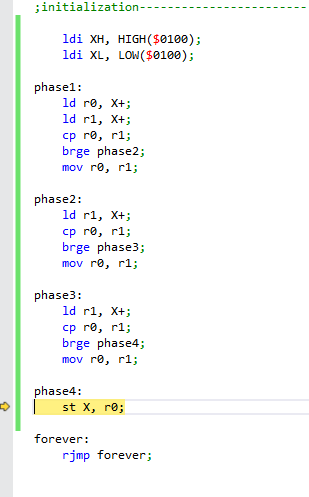
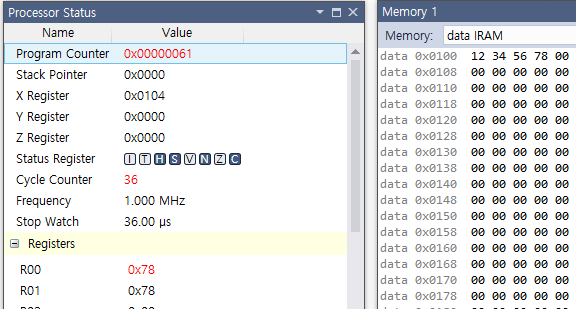
첫 번째 루프에서 r0에 100번지의 값을 할당, r1에는 101번지의 값을 할당한 뒤 두 값을 비교한다. 만약 r0, 즉 100번지 값이 크다면 바로 두 번째 루프로 넘어가고 아니라면 r0에 r1값을 저장한다.

phase2 루프를 수행하고 난 뒤 (r0이 r1보다 작으므로 mov 명령어 수행)

마찬가지로 두 번째 루프에서 r1에 102번지의 값을 할당, r0과 r1, 두 값을 비교한다.

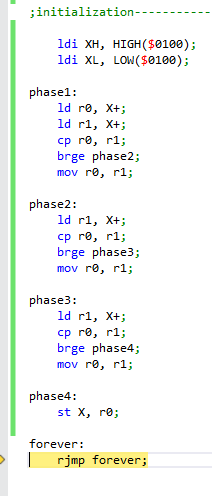
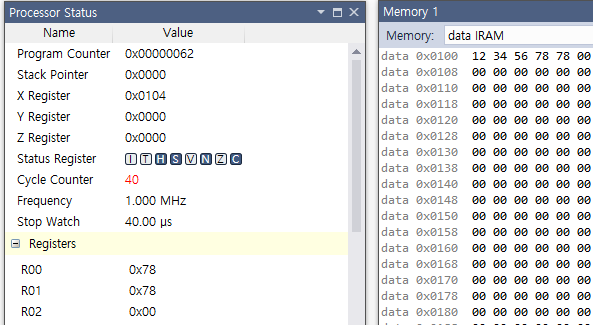
만약 r0값이 크다면 바로 세 번째 루프로 넘어가고 아니라면 r0에 r1값을 저장한다.

phase3 루프를 수행하고 난 뒤 (r0이 r1보다 작으므로 mov 명령어 수행)

세 번째 루프에서 r1에 103번지의 값을 할당, r0과 r1, 두 값을 비교한다.

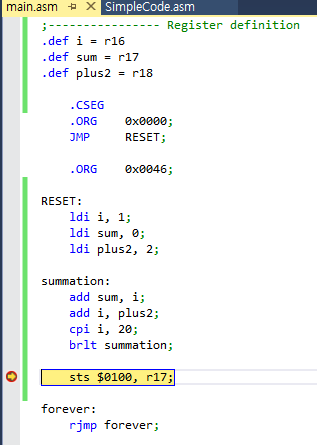
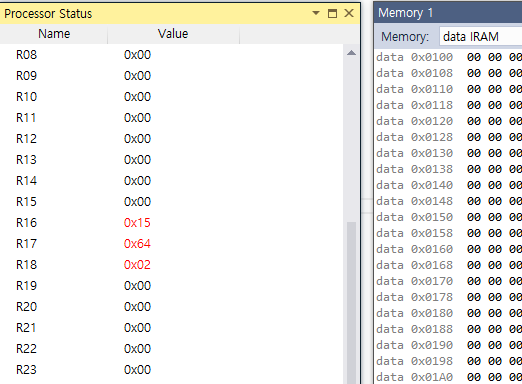
만약 r0값이 크다면 바로 네 번째 루프로 넘어가고 아니라면 r0에 r1값을 저장한다.

final 루프를 수행하고 난 뒤 데이터영역의 0x0104에 가장 큰 값이 저장됨을 확인할 수 있다.

네 번째 루프에선 비교를 통해 남게된 가장 큰 r0값을 104번지에 저장한다.

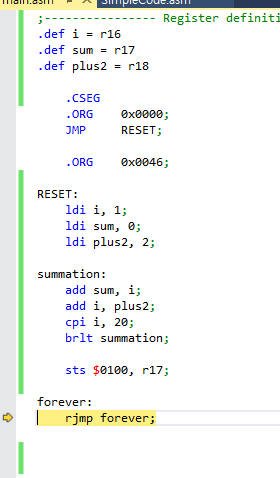
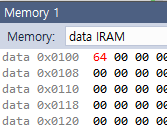
Lab 3-3

r16이 20이 넘는 순간 조건분기문에 의해 summation 루프를 더 이상 반복하지 않는다. sum 값은 r17레지스터에 저장되어있다.

r16은 계속해서 증가되는 홀수를 의미하고 r18은 홀수를 증가시키기 위한 +2를 의미한다.

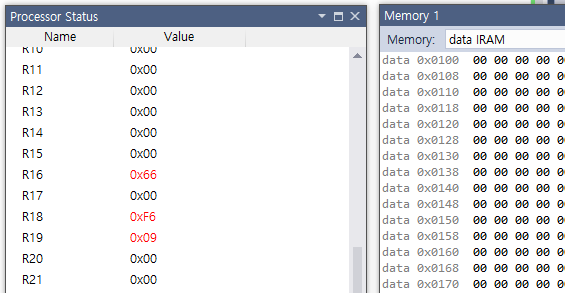
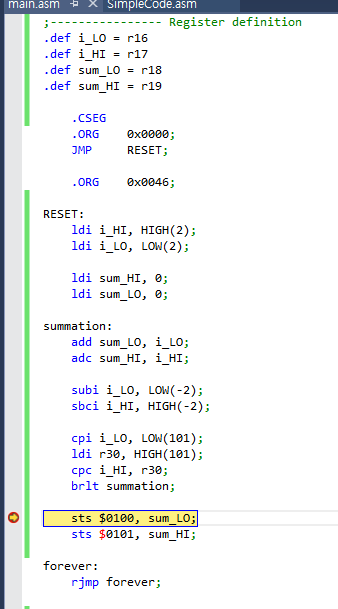
sum(r17)에 i(r16)값을 더하고 i값은 2씩 증가시킨다, 이때 cpi를 통해 i와 20을 비교, 만약 i값이 20을 초과하면 summation 루프의 반복을 빠져나간다.

데이터의 0x0100 번지에 sum 값이 저장된다.

루프의 반복을 빠져나가면, 즉 i가 20을 넘어가면 sts를 통해 100번지에 sum값을 저장한다.

Lab 3-4



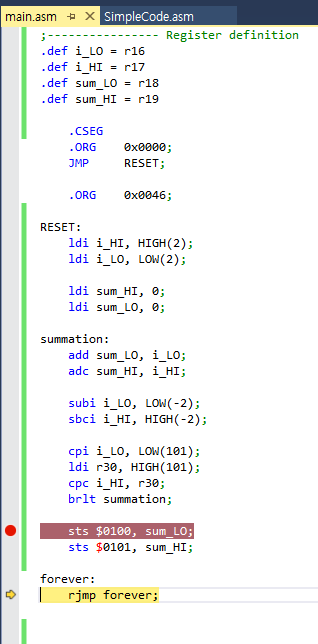
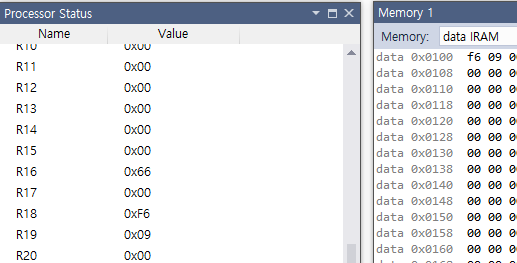
16비트로 나누어서 i를 설정해주고 i값을 100이하인 범위에서 2씩 올려가며 summation을 하였다.

16비트 덧셈을 진행해야 하기 때문에 증가되는 짝수는 i\_HI, i\_LO 두 부분으로 나누어 주었으며

마찬가지로 짝수의 총합도 sum\_HI, sum\_LO 로 두개의 레지스터를 이용한다. subi 를 이용해 i에 2씩 더해주었으며 adc를 사용하였기 때문에 16비트 덧셈을 의미한다.

sum 역시 sum에 i를 add, adc를 이용해 계속해서 더해주었으며 16비트 덧셈을 의미한다.

조건분기문을 이용해 i 값이 101을 넘어가면 summation 루프를 빠져나가도록 설정하였다.

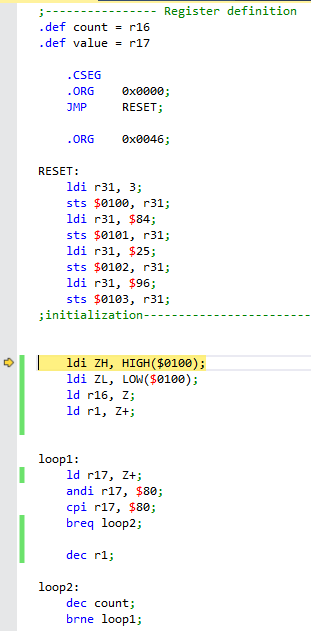
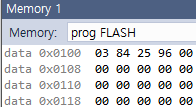
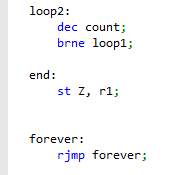
 

데이터 영역에 데이터가 저장되었음을 확인할 수 있다.

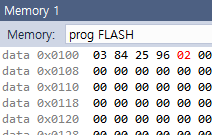
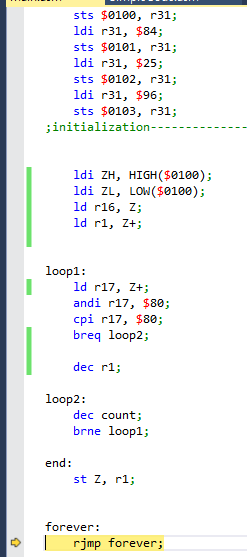
모든 동작이 끝나고 summation 루프를 빠져나가면 sts 명령어를 통해 sum 값이 100번지와 100번지에 저장된다. 100번지가 LOW이고 101번지가 HIGH이다.

0x09F6 = 2550 이므로 제대로 데이터가 저장되었다.

Lab 3-5

주어진 가정으로 초기화를 해준 모습이다.



0x0100 번지에서 받은 count 수 를 이용하여 데이터들을 count수만큼 검사하고 검사가 끝나면 바로 그다음 번지에 negative 값의 개수를 저장한다.

r1과 r16(count)에 데이터 개수를 저장한다.

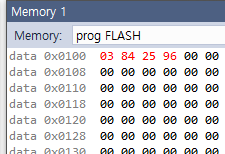
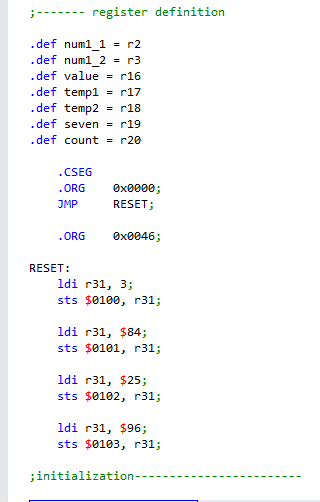
첫번째 루프에선 r17에 첫번째 정보값을 저장하고 그 값을 0x80과 and masking을 수행한다.

만약 정보값이 음수여서 and masking을 수행한 뒤 0x80과 값이 같다면 바로 loop2로 가고 아니면 r1값을 감소시킨다. (음수가 아니면 감소)

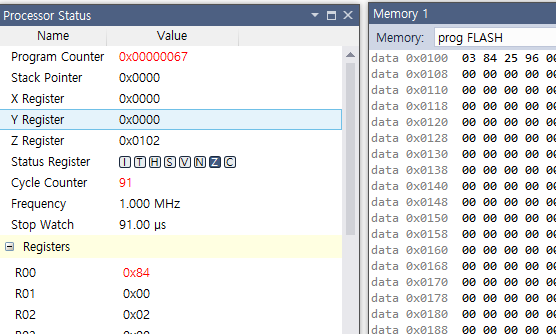
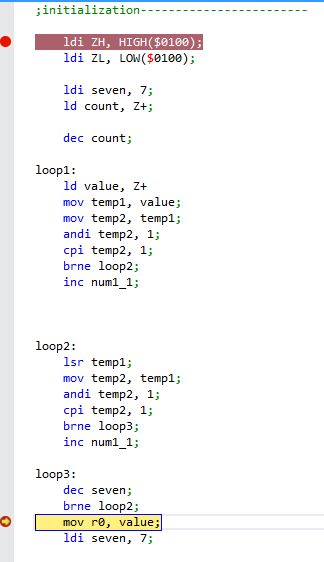
두번째 루프에선 count 값을 감소시키며 count개수만큼 첫번째루프부터 다시 반복하도록 한다.

count값이 0이되어 두번째루프를 빠져나가면 입력받은 정보주소들의 끝에 남은 r1값 (음수값)을 저장한다.

Lab 3-6



주어진 가정으로 값들을 설정해주었다.



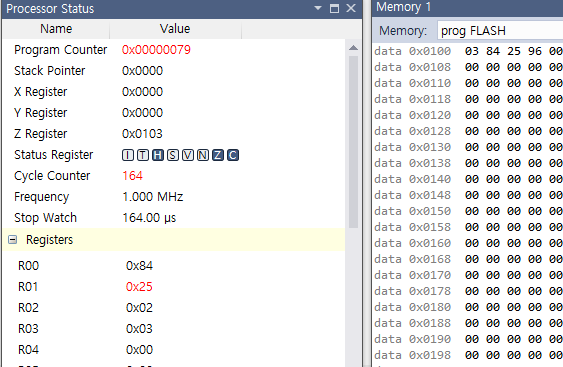
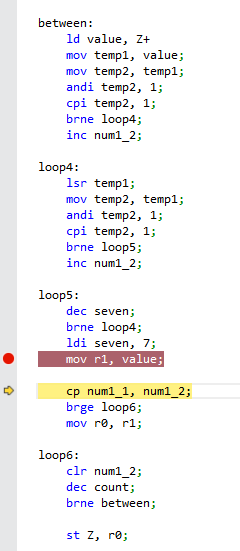
첫 번째 정보의 1의 개수를 카운트하고 r02에 저장, 첫 번째 정보값은 r0에 저장하였다.

7을 나타내는 seven레지스터와 정보의 개수를 나타내는 count레지스터를 설정해준다. 나중에 필요한 실제 반복회수는 count 보다 1작으므로 count -1을 수행하였다.

첫번째루프는 value 레지스터에 첫번째 정보값을 저장하고, 1과 and masking을 한다.

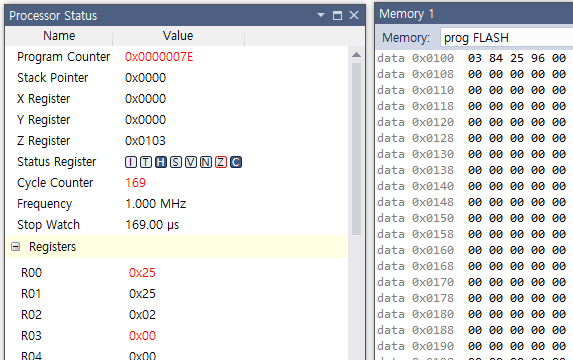
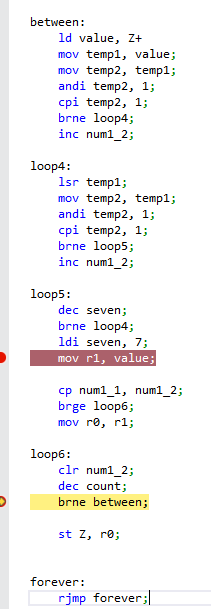
두번째루프와 세번째루프는 value 레지스터 정보값을 temp1과 temp2를 이용해 저장 후 lsr과 조건분기문을 이용해 각 자리수의 1들을 모두 세어준다. 이때 num1\_1레지스터에 첫번째 정보값의 1의 개수를 저장한다.

마지막으로 r0에 첫번째 정보값을 저장한다.



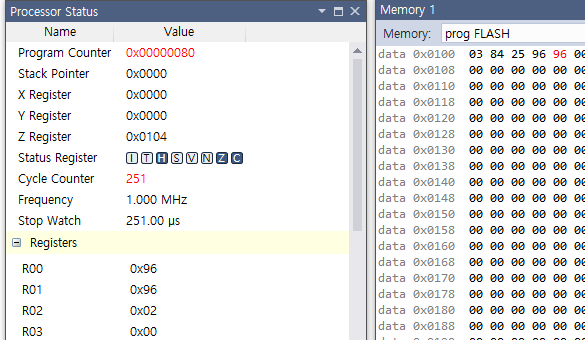
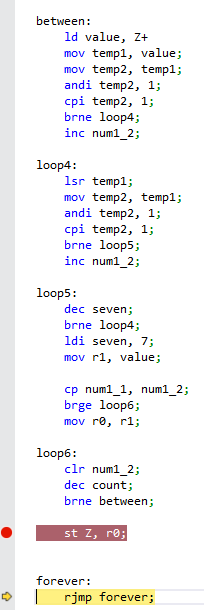
비교하려는 그 다음 정보의 1의 개수를 r3에 저장하고 정보값은 r1에 저장하였다.

between 루프와 4번째 루프는 이전 루프와 같이 입력받은 두번째 정보값의 1의 개수를 세준 뒤 num1\_2 레지스터에 저장한다. 그 뒤 r1에 두번째 정보값을 저장해준다.



첫 번째 값의 1의 개수와 그 다음 정보의 1의 개수를 비교한뒤 1의 개수가 많은 값을 r0에 저장한다. 그 뒤 남아있는 정보의 수만큼 동작을 반복한다.

num1\_1 , num1\_2 를 비교하여 첫 번째 정보와 그 다음번 정보중 1의 개수가 많은 정보를 r0에 저장하도록한다. 그 뒤 남아있는 정보들을 모두 위와 같은 방식을 반복해준다.



모든 정보들의 비교를 끝낸 뒤에 1의 개수가 가장많은 정보가 정보들의 마지막에 저장된다.

고찰

다양한 방식으로 조건분기문을 이용하여 루프를 구성, 이용하는 방법을 이번 과제를 통해 익혔다. 조건 분기문을 이용하여 원하는 루프로 이동할 수 있고, 만약 조건분기문이 없다면 루프는 위에위부터 아래로 차례차례 디버깅됨을 알았다.

또한 데이터 영역의 값들을 순차적으로 비교하는 작업을 진행할 때 X, Y , Z 레지스터의 주소를 저장하고 후위연산자를 통해 순차적인 데이터의 접근과 이용이 가능함을 실습을 통해 다시한번 복습하였다.

조건분기문을 이용하여 원하는 상황에 원하는 루프로 접근하는 것은 위력적이고 유용하지만, 아직 함수를 배우지 않아서인지 함수를 구성하고, 함수를 이용하는 방법을 알게된다면 루프컨트롤 보다 더욱 유용할 것이라고 생각된다.