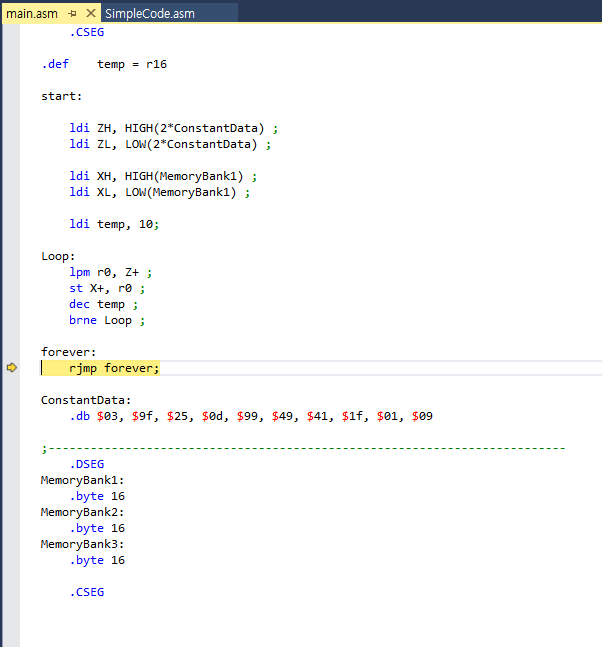
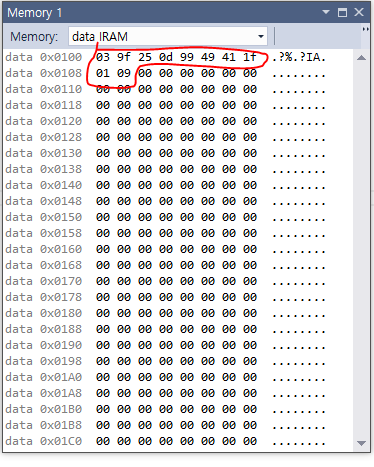
프로그래밍, 소스파일, 데이터메모리뱅크의 데이터부분, 고찰에 대해 제출

#2-1





고찰

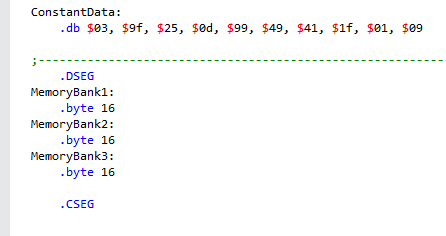
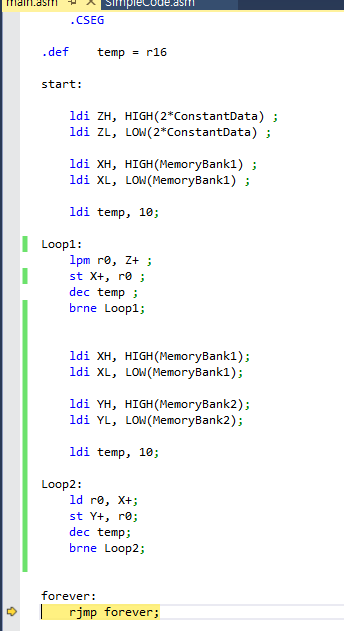
ConstantData 를 프로그램메모리영역에 설정한다. 프로그램메모리영역의 데이터들은 read only 메모리 이므로 Constant 상수라 할 수 있다. 프로그램메모리의 ConstantData의 주소를 Z레지스터의 저장하기 위해 ldi ZH, High(2\*ConstantData); ldi ZL, LOW(2\*ConstantData) ; 명령어를 사용하였다. 이를 통해 Z레지스터의 저장하였으며 이때 주목할 점은 프로그램메모리는 원하는 주소에 접근하기 위해 2를 곱해주어야 한다는 점인데 그 이유는 프로그램메모리의 데이터크기는 16비트(2바이트)이기 때문이다. 또한 위와 같은 방식으로 X레지스터에는 MemoryBank1의 주소를 할당하였다.

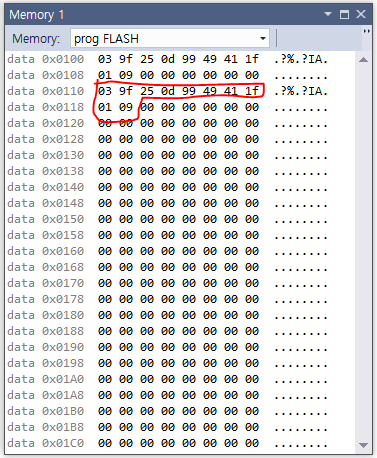
저장되는 주소는 0x0100이다. 또한 Loop문은 brne Loop를 통해 이전 계산에서 Zero flag가 실행되기 전까지 Loop를 반복하도록 설정해준다. 이는 temp (r16레지스터) 변수값을 설정해주고

dec temp 를 통해 temp 변수값을 1씩줄여주어 탈출조건을 설정해준다. 또한 Loop내 반복을 통해

Z주소의 데이터 (ConstantData) 를 X주소의 데이터로 이동시킨다. 이 방법은 lpm r0, Z+ 을 통해 Z주소의 데이터 값을 r0레지스터로 옮기고, st X+, r0을 통해 r0레지스터값을 X주소의 데이터값에 저장한다. 후위증감자를 통해 (+) 연산 후 X, Z레지스터가 가리키는 주소값을 증가시켜 모든 배열값들을 옮기도록 해준다. 프로그램코드뿐 아니라 데이터코드를(.DSEG) 통해 MemoryBank들의 크기 설정해줘야 한다.

#2-2



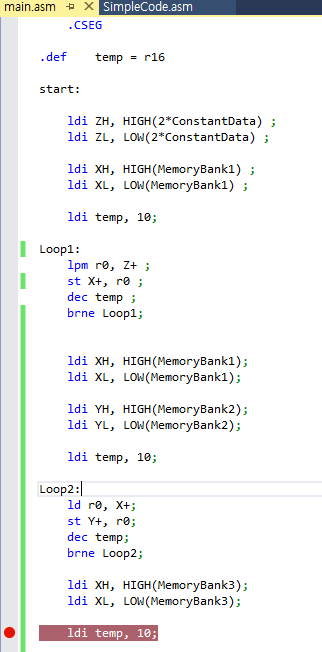


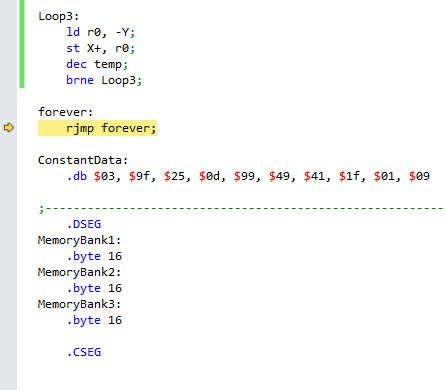
고찰

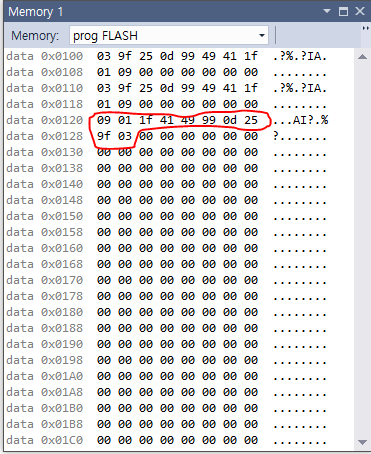
X레지스터에 저장된 주소값이 Loop1의 X+를 통해 증가하였으므로 X레지스터의 주소값을 다시 MemoryBank1으로 초기화시켜준다. 또한 Y레지스터에 MemoryBank2 주소를 저장해주고 Loop2의 반복을 위해 temp 변수를 다시 10으로 재설정해준다. 그 뒤 Loop2의 동작은 Loop1과 유사하다. 다만 Loop1은 프로그램메모리의 데이터를 레지스터로 가져왔기 때문에 lpm 명령어를 사용하였지만 Loop2는 데이터 메모리에서 데이터를 레지스터로 가져왔기 때문에 ld 명령어를 사용한다는 차이점이 있다. 다음의 동작을 모두 마쳐 Loop2를 빠져나가면 MemoryBank2에

MemoryBank1 값들이 차례로 할당되어있음을 확인할 수 있다.

#2-3





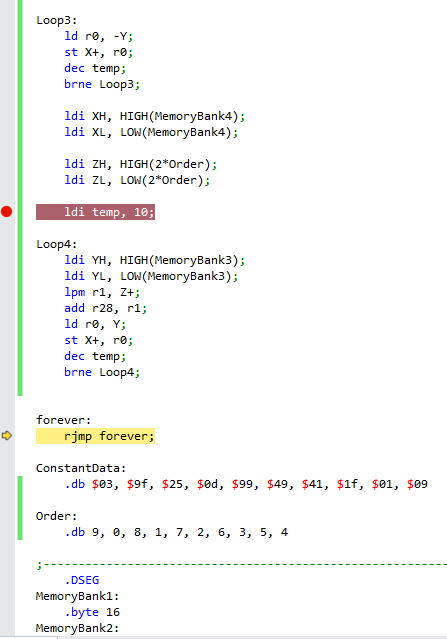
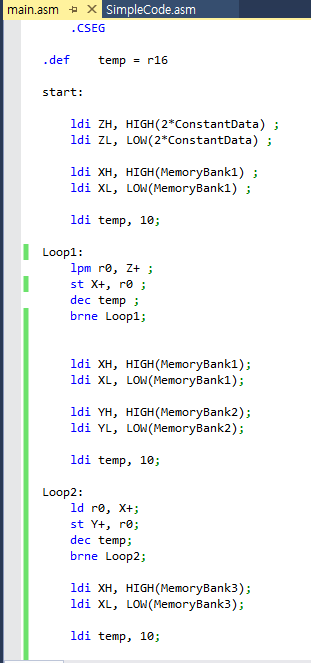


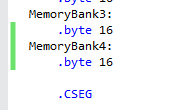
고찰

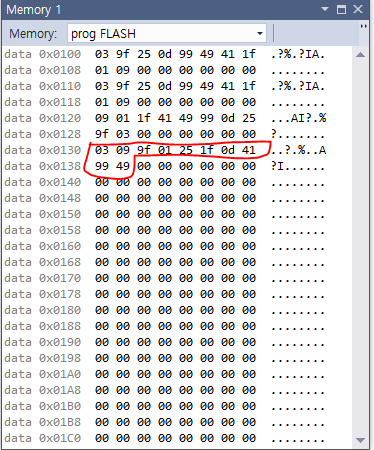
MemoryBank3에 MemoryBank2의 데이터를 거꾸로 넣어주기 위해서 우선 X레지스터에

MemoryBank3의 주소를 할당하였다. 그 뒤 temp 변수를 10으로 재설정하였고 Loop3에서 -Y, 즉 전위감소 연산자를 이용하여 Y의 주소값이 MemoryBank2의 마지막 데이터를 가리키도록 설정 후 그 데이터값을 r0에 저장하였고, 이를 X의 주소값이 가리키는 MemoryBank3의 첫 부분부터 차례대로 넣어주었다. 주의할점은 -Y를 사용한 부분인데 이는 Loop2를 끝내고 나면 Y가 가리키는 주소가 MemoryBank2를 벗어나기 때문에 전위감소연산자를 사용한 것이다.

#2-4







고찰

MemoryBank3에 있는 메모리를 Order배열에 적힌 offset을 주어 MemoryBank4로 이동시키기 위하여 우선 X레지스터에 MemoryBank4 주소를 할당하였다. 그 뒤 Z레지스터에 Order배열의 주소를 할당하였고 (프로그램 메모리이므로 주소에 2를 곱해준다.) temp 변수에 10을 주어 반복조건을 설정하였다. Loop4 내에서 Y 레지스터에 MemoryBank3의 주소를 할당해주었다. 이는 Loop4가 반복될 때 마다 Y레지스터 값을 MemoryBank3의 첫 주소값으로 초기화해줌을 의미한다. 그 후 Z레지스터의 offset 값을 r1에 저장하고 r28, 즉 Y레지스터의 low 부분에 offset을 더하여 배열의 인덱스 접근과 같은 효과를 주었다. 그렇게 변경한 Y주소가 가리키는 데이터값을 r0에 저장 후 X주소가 가리키는 배열에 옮겨주었다.

전반적으로 C언어와 비슷한 듯 다른 어셈블리어는 low level 언어여서 그런지 C언어에서의 단순한 동작들을 조금은 돌아간다는 느낌이 있었다. 그러나 거의 비슷한 동작을 기계어 수준에서 쉽게 사용가능 하다는 점이 큰 장점인 것 같다.