어셈블리프로그램 설계및실습 보고서

과제 주차: 2주차

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화요일 6, 7

학 번: 2019202050

성 명: 이강현

제 출 일: 2022.09.30(금)

1. Problem Statement

   flag를 활용하여 조건문을 사용해보고 데이터 흐름에 대해 이해한다.

   LDR와 STR을 통해 데이터가 레지스터와 메모리에서 어떻게 이동하는지 알고 condition execution을 적절하게 사용하며 원하는 처리를 할 수 있게끔 한다. 메모리의 저장방식인 리틀엔디안과 빅엔디안의 차이를 알고 이를 활용해본다.

2. Design

   <Problem1>

   메모리에 값을 할당하고 이를 1바이트 단위로 불러와 상수 10과 비교하여 크면 1 작으면 2 같으면 3을 R5에 넣는 문제이다.

   흐름은 다음과 같다.

   우선 TEMPADDR1변수에 1000이라는 메모리 주소값을 넣어준 후 r4에서 TEMPADDR1 변수 값을 로드 하였다. 임의의 값 10보다 작은 7, 10과 같은 10, 10보다 큰 17을 해당 주소가 1000인 메모리에 1바이트 단위로 넣어주었다. 그 후 r4에서 1바이트씩 읽어오기 위해 LDRB명령어를 사용하였고 각각 값들을 r0,r1,r2에 넣어주었다.

   CMP명령어를 통해 각각의 값들을 비교한 후 flag에 따라 레지스터에 저장하기 위해 위에서부터 차례대로 MOVGT, MOVMI,MOVEQ 명령어를 사용하였다.

   <problem 2>

   r1,r2,r3,r4에 각각 1,2,3,4를 저장하고 r5,r6에 각각 0x04030201과 0x01020304가 저장되게끔 구현한다.

   메모리의 저장방식을 이해할 필요가 있는 예제이다.

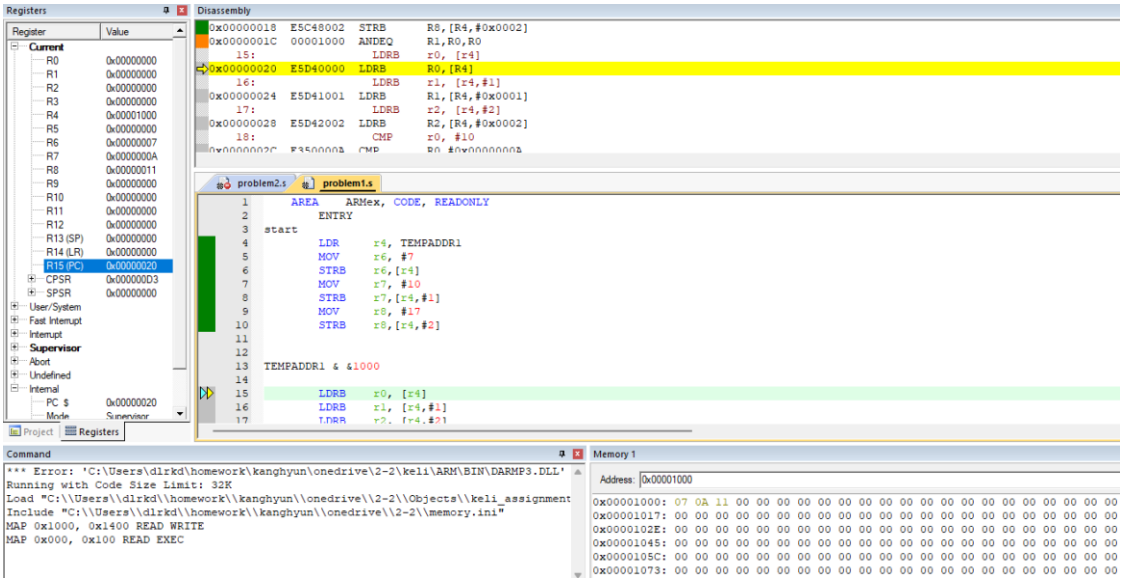
   Little-endian방식으로 메모리가 데이터를 저장하므로 r5와 같은 값을 저장하려면 메모리에는 01 02 03 04순으로 저장되어야 하고 r6와 같은 값을 저

장하려면 04 03 02 01순으로 저장되어야 한다.

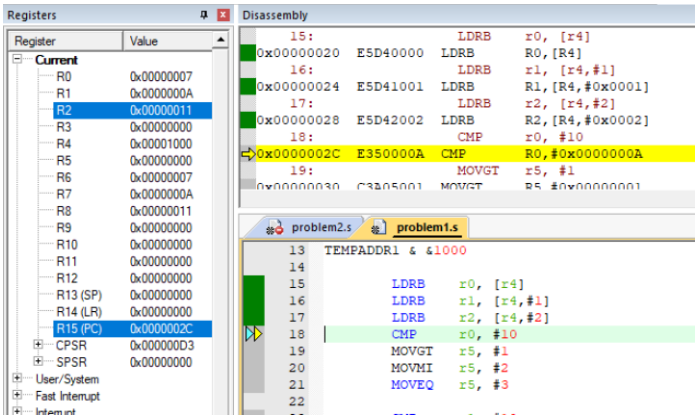
따라서 MOV명령어를 통해 r1,r2,r3,r4값들을 세팅해준 뒤 STRB명령어를 통해 1바이트씩 이동하며 순차적으로 저장한다. 이때 메모리의 값은 r4에 저장해두었고 이를 활용하여 메모리에 접근할때는 Pre-index addressing 방법을 사용하여 r4의 값이 변하지 않으면서 각각의 레지스터가 메모리에 한바이트씩 이동하여 접근할 수 있게 구현하였다.

3. Conclusion

<problem1>

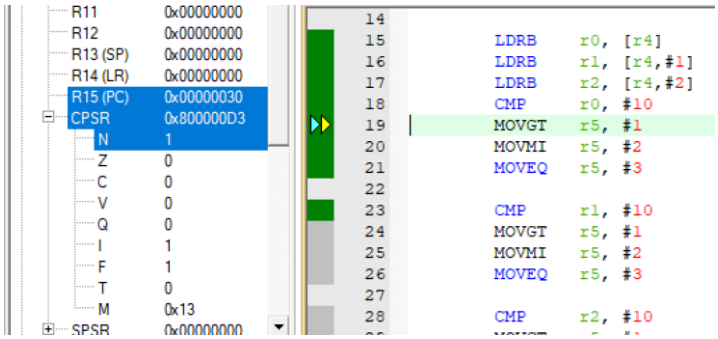


LDRB명령어로 데이터를 불러오기 전 문제의 초기상태를 마련해놓은 상태이다. 0x00001000부분에 07 0A 11값이 잘 들어가 있는 것을 확인할 수 있다.

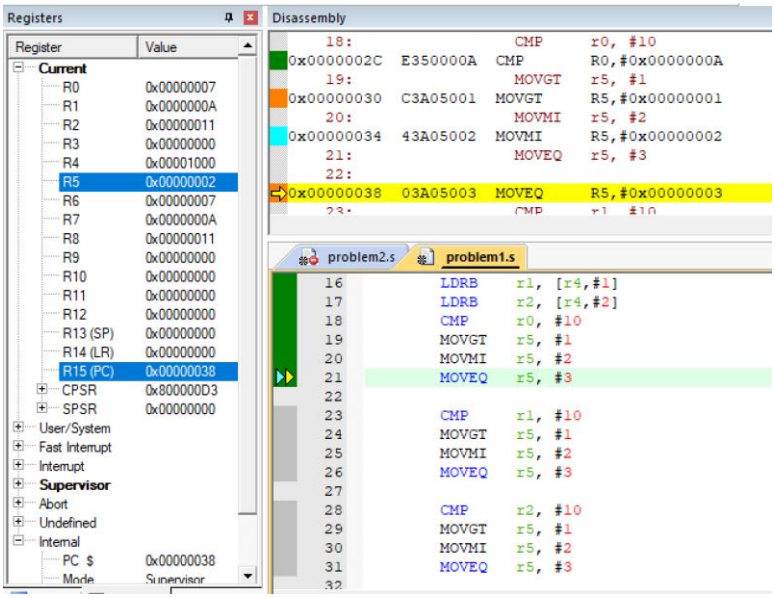


데이터가 잘 불러왔는지를 확인하기 위해 메모리에 저장하기위해 사용했

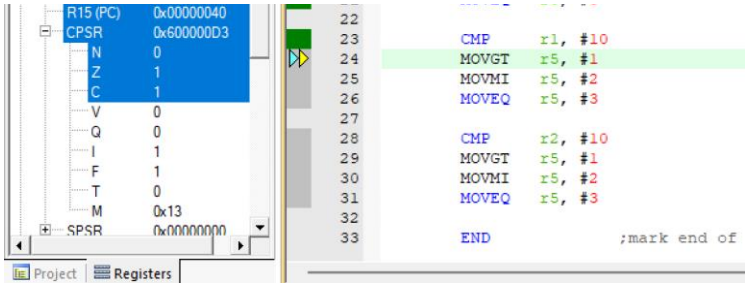
던 r0,r1r2가 아닌 r6,r7,r8에 값을 load하였다.



CMP명령어를 사용하여 r0(7)을 10과 비교한 결과 CPSR을 보면 앞자리가 8로 바뀜을 알 수 있다. NZCV가 1000으로 저장되었기 때문에 8을 나타냈고 이는 CMP결과 flag N이 1 즉 7-10이 음수이므로 negative임을 알 수 있다.

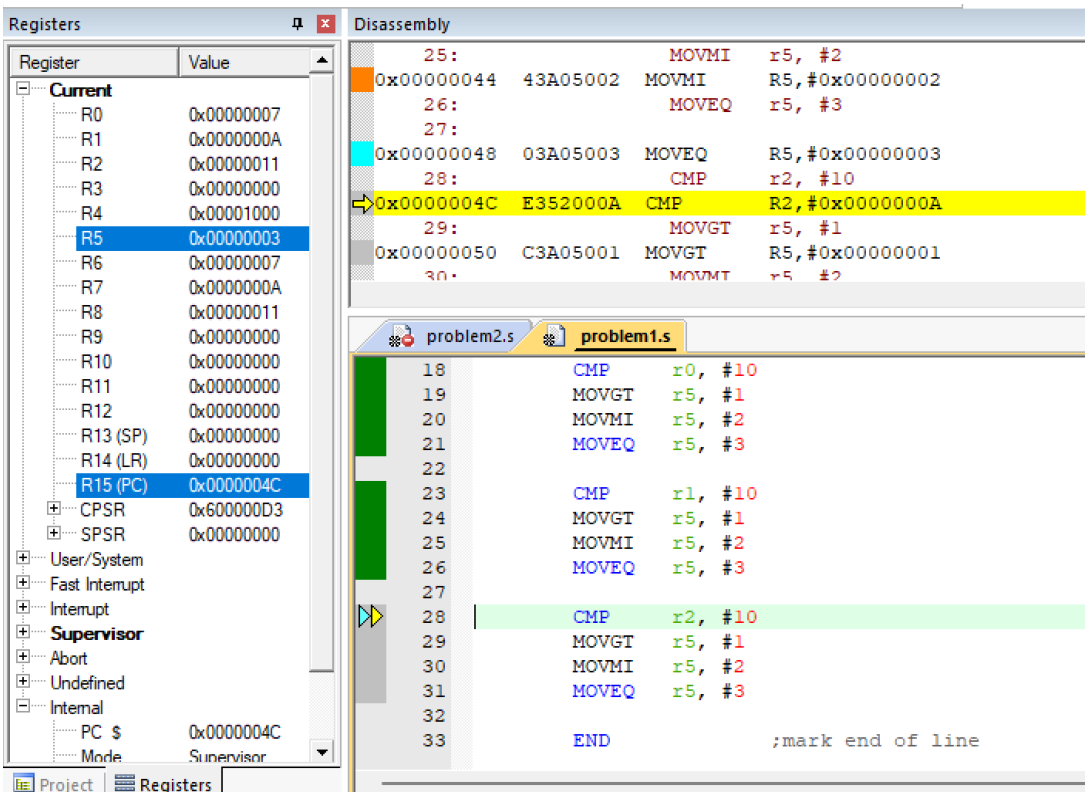


따라서 위 화면처럼 MOVMI일 때 문장이 실행되어 r5에 2가 대입된 것을 확인할 수 있다.

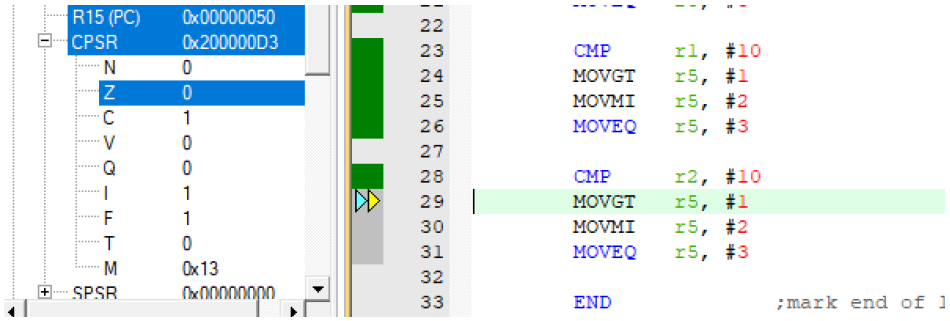


마찬가지로 r1(10)을 10과 비교했을 때는 flag Z가 1이되었고 C가 1이 되

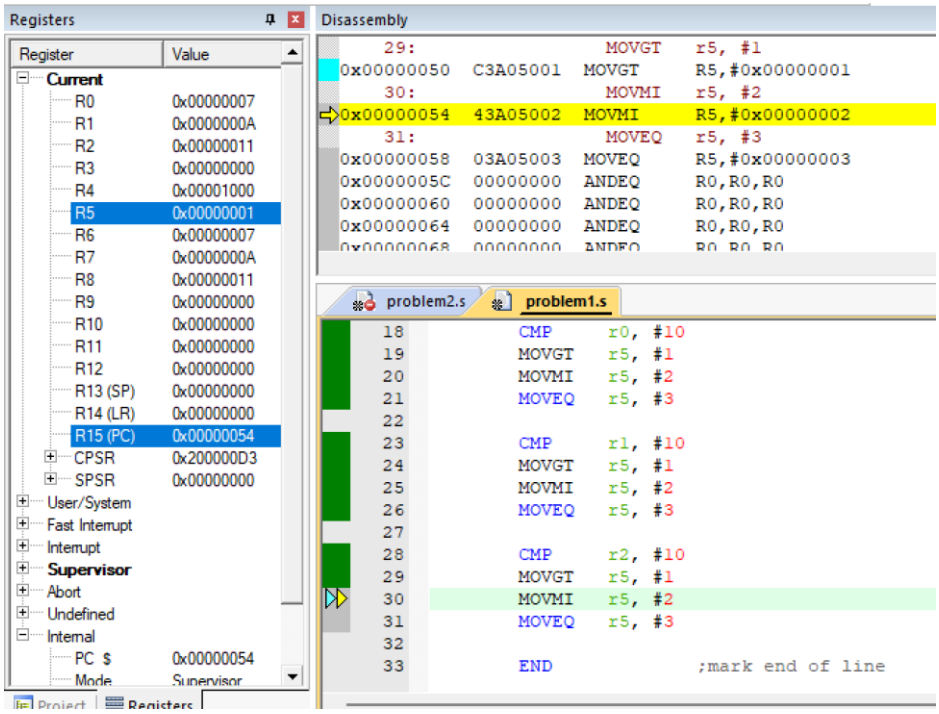
었다. 이는 10을 나타내는 1010과 10의 보수인 0110을 더했기 때문에 Carry는 발생했으나 올바른 값이 도출되었으므로 오버플로우를 나타내는 flag인 V의 값은 바뀌지 않은 것을 확인할 수 있다.



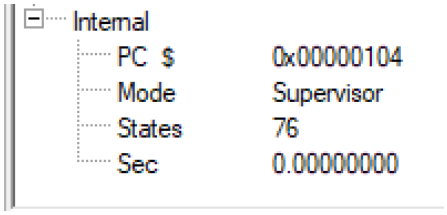
따라서 위와 같이 Z가 set상태일 때 작동하는 MOVEQ문장이 실행되어 3이 r5에 할당되었음을 확인할 수 있다.



마지막으로 r2(17)을 10과 비교한 결과 전에 1이었던 Z flag가 비교결과값이 양수이므로 0으로 바뀜을 확인할 수 있고

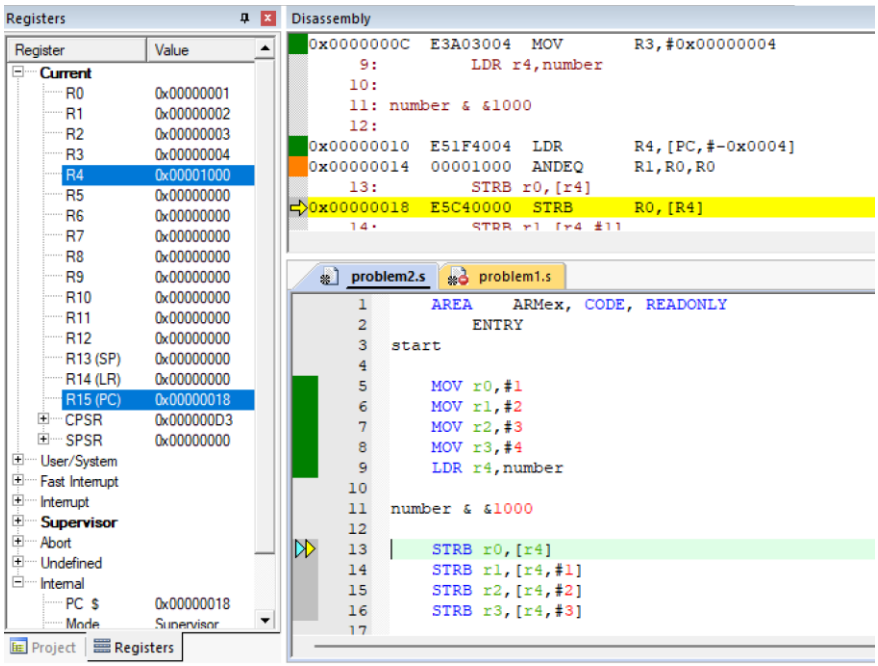


위와 같이 Z가 clear 상태이고 N과 V가 같은 상태일 때 작동하는 MOVGT 문장이 실행되어 r5에 1이 할당됨을 알 수 있다.

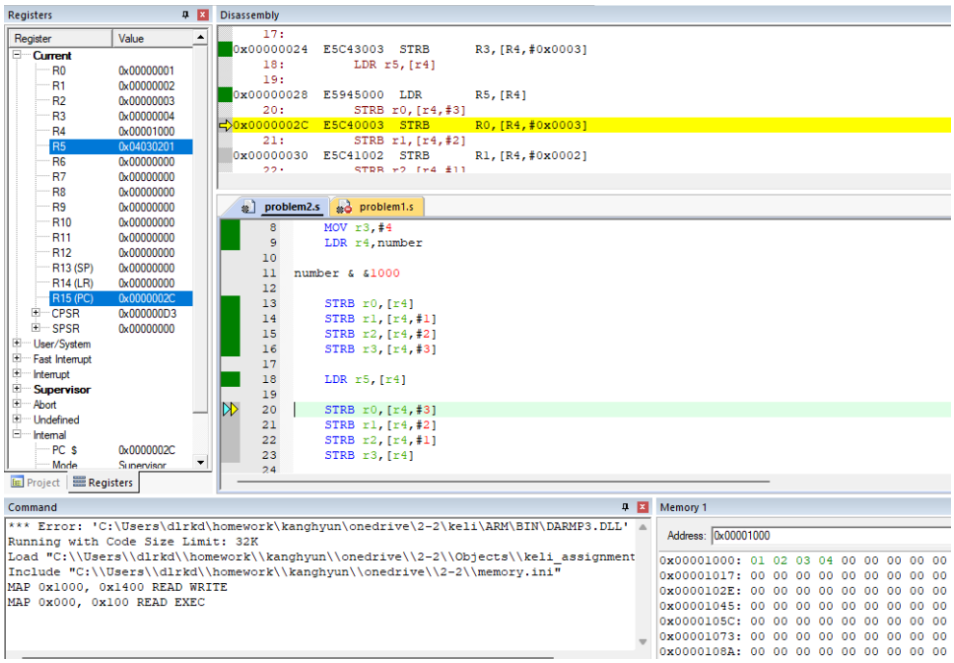


총 76번의 states을 가지는 코드임을 확인한다.

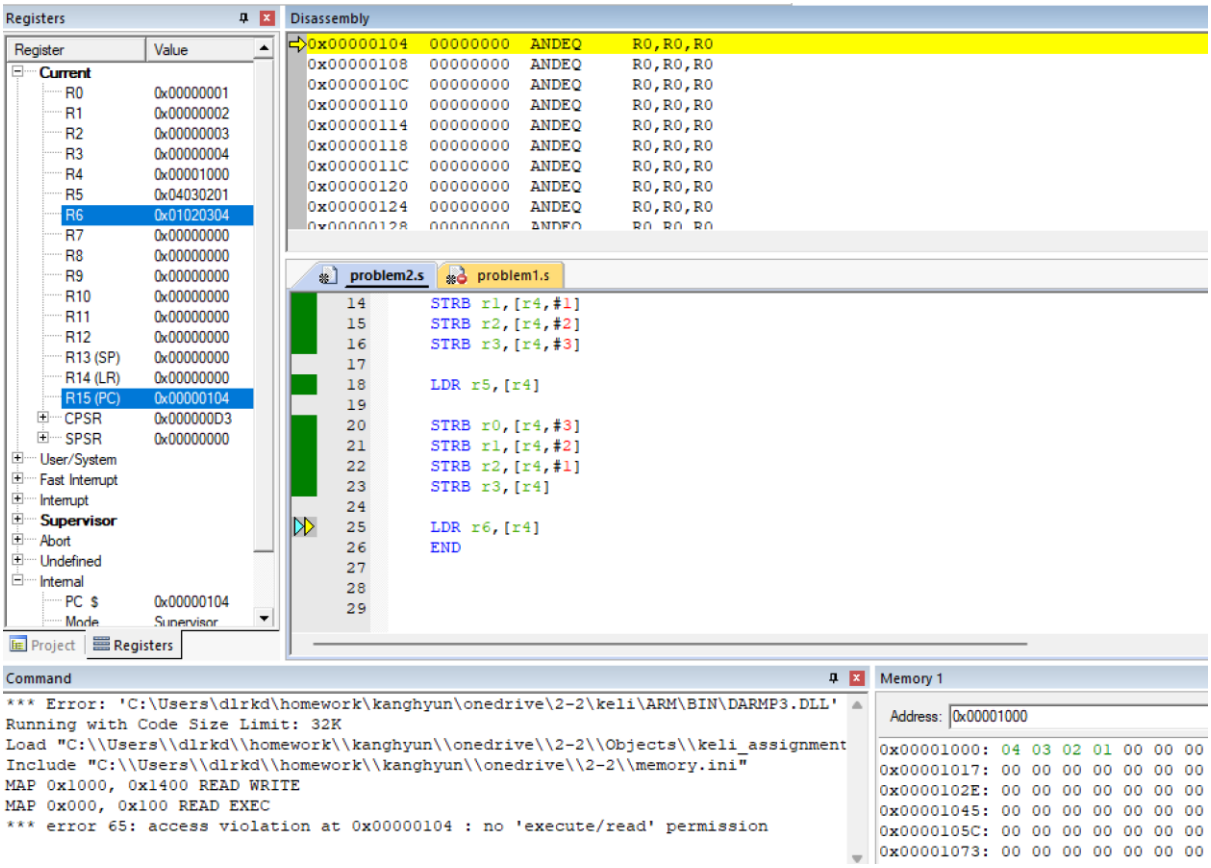
<problem 2>



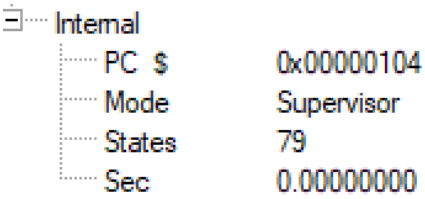
레지스터값 초기 저장상태이다. 문제 배경에 맞게 구현하였다.



Pre-indexing address 방식으로 r4의 메모리에 한 바이트단위로 순차적인 접근을 하기 위해 1,2,3이라는 offset을 이용했다. 그 결과 little-endian방식이므로 LDR 명령어 문장 이후 r5에 04030201이 저장됨을 알 수 있다.



같은 방식으로 3,2,1 순서로 offset을 적용하여 r4의 메모리에 값을 저장하였고 메모리를 확인하면 순서가 바뀐 것을 확인할 수 있다. 이를 통해 r6에는 01020304를 저장할 수 있었다.



총 79번의 states가 나왔다.

4. Consideration

어셈블리언어는 단순히 데이터를 저장하는 것이 좀 더 상위 레벨의 언어보다 구체적이고 신경써야하는 부분이 많았다. if문으로 조건문을 보기쉽게 사용하거나 =을 통해 값을 단순히 할당해주는 언어와는 달리 flag를 사용하여 논리를 이해하기 어렵고 저장할 메모리를 설정해주어야 하는 것과

little-endian 방식인점은 직관적으로 이해하기가 어려웠다.

이번 예제를 통해 데이터가 어떻게 흘러가는지 저장은 어떻게 하는지 불러오는 것은 어떻게 하는지를 알았고 바이트 단위로 저장하는 법, 비교연산을 통해 condition execution을 사용하는 법등등 기초적인 개념을 학습하는데 도움이 되었던 것 같다.

5. Reference

이형근/어셈블리프로그램 설계 및 실습/광운대학교(컴퓨터정보공학부)/2022