1. 배경지식

어셈블리어는 다른 프로그래밍 언어보다 기계어에 더 가까운 Low-level language이고, 기계어와 1 대 1 대응하는 구조를 가지는 것이 특징이다. 다른 프로그래밍 언어와 비교해서 공간적, 시간적 효율이 높아 실시간으로 정보를 처리하는 영역에 이용된다. 반면 다른 High-Level language보다 인간이 이해하기 어렵기 때문에 프로그램의 작성과 유지보수가 어려우며, 이식성이 좋지 않다는 단점이 있다.

어셈블리 코드는 수행할 동작을 의미하는 Opcode와, 해당 동작에 필요한 값이나 레지스터 등을 의미하는 Operands 부분으로 구성된다. Opcode는 수행할 동작이 축약어 형태로 간단하게 나타난다. Operand 부분에 사용되는 레지스터는 CPU내부에 존재하는 저장공간이다. 마찬가지로 값을 저장하는 역할을 하는 메모리는 CPU외부에 있다는 차이점이 있다. 또한 레지스터의 크기가 더 작고, 속도는 더 빠르다.

어셈블리에서는 CMP명령어를 이용해 레지스터에 저장된 두 값을 비교할 수 있다. **CMP r0, r1**과 같은 형태로 이용하고, r0에 저장된 값에서 r1에 저장된 값을 빼는 연산을 한 다음, 연산의 결과에 따라 CPSR의 flag 값을 변경한다. CPSR은 Current Program Status Register의 약자이다. CPSR 내부에는 다양한 flag가 있다. CMP의 결과로 N,Z,C,V flag값이 변경된다. N은 r0-r1의 결과가 음수임을 의미한다. 즉 r0>r1 인 경우이다. Z는 연산의 결과가 0인 경우이다. 이 경우 r0와 r1에 저장된 값이 같음을 의미한다. C는 연산 도중 Carry가, V는 Overflow가 일어났음을 의미한다. [i]

어셈블리에서 조건부 실행 명령은 조건이 충족되었을 때 실행할 명령어 뒤에 해당 조건을 나타내는 flag를 표시하여 작성한다. 과제를 진행하면서 사용한 flag는 GT, MI, EQ이다. GT flag는 Signed >=인 경우, MI는 <인 경우, EQ는 비교한 두 레지스터에 저장된 값이 같은 경우에 해당한다.

레지스터에 저장된 값을 메모리에 저장하는 명령어는 STR이고, 반대로 메모리에 저장된 값을 레지스터에 저장하는 명령어는 LDR이다. 각 명령어의 뒤에 저장할 값의 바이트 수를 지정할 수 있다. 메모리에 값을 저장할 때는 Little-Endian 방식으로 저장된다. Little-Endian은 값의 가장 뒤에 위치하는 LSB가 가장 먼저 저장되는 방식이다. 결국 레지스터에 저장된 값의 순서가 거꾸로 저장된다. 이와 반대되는 것은 big-endian으로, 이 경우 레지스터에 저장된 순서대로 값이 저장된다. [ii]

2. 코드내용

(1) Exercise1.s

메모리에 저장된 수를 읽어 10과 비교한 뒤, 각각의 경우에 따라 r5에 다른 값을 저장하는 프로그램이다. 프로그램은 메모리에 값을 저장하는 부분이 가장 먼저 나오고, 값을 읽어 10과 비교한

뒤, 결과에 따라 다른 동작을 실행하는 부분이 세 번 반복된다. 프로그램이 실행되면, 레지스터에 저장된 값을 1바이트씩 메모리에 저장한다. 메모리의 주소를 초기값으로 되돌리고, 메모리에서 한 바이트를 읽어 r4에 저장한다. 10을 저장해 둔 r6과 r4를 CMP명령어를 이용해 비교하여 CPSR의 flag를 변경시킨다. 이후 flag값에 따라 메모리에서 읽은 값이 10보다 큰 경우 1, 작은 경우 2, 같은 경우 3을 r5에 저장한다.

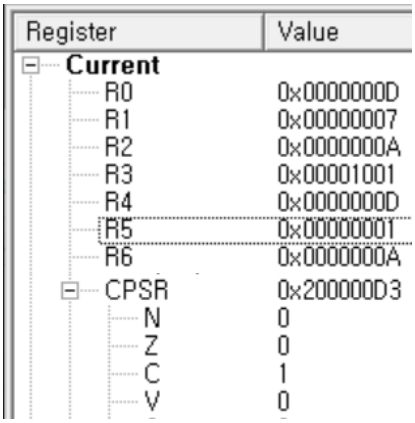
(2) Exercise2.s

레지스터에 저장된 값을 순서대로, 반대 순서로 다른 레지스터에 저장하는 문제이다. 프로그램은 레지스터에 수를 저장하는 부분이 가장 먼저 나오고, r5에 역순으로 값을 저장하기 위해 메모리에 값을 1바이트씩 저장한다. 이후 메모리 주소를 초기값으로 되돌리고 한 번에 값을 읽어 r5에 저장한다. r6에 순서대로 값을 저장하기 위해서 메모리에 반대순서로 1바이트씩 값을 저장한다. 이후 메모리 주소를 초기값으로 되돌리고, 한 번에 값을 읽어 r6에 저장한다.

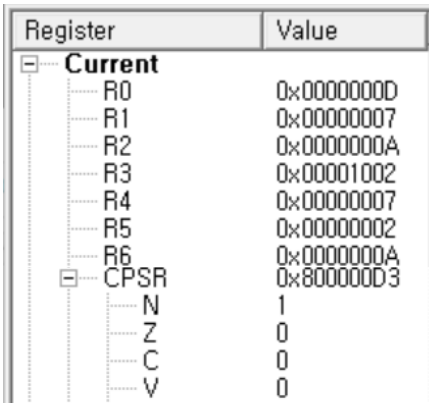
3. 결과

(1) Exercise1.s

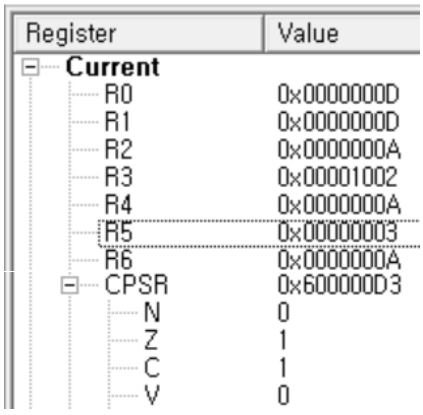
1. r0의 값을 비교한 후



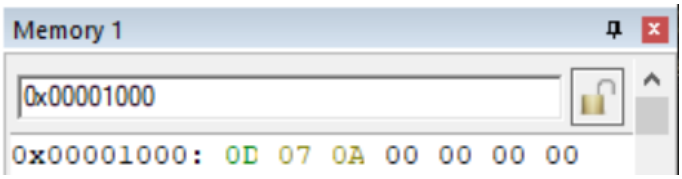
2. r1의 값을 비교한 후



3. r2의 값을 비교한 후

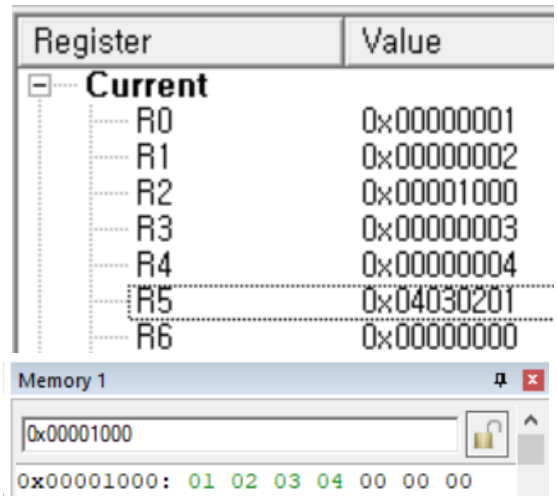


4. 메모리

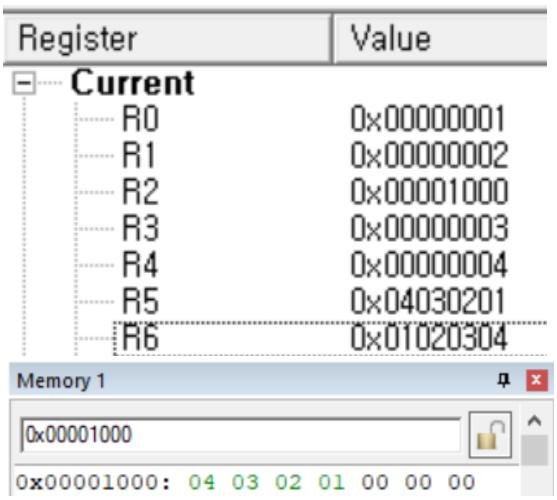


(2) Exercise2.s

1. r5에 값을 저장한 뒤



2. r6에 값을 저장한 뒤



4. 고찰(결과에 대한 설명 필수)

(1) Exercise1

메모리에 13,7,10의 값을 넣어서 프로그램을 작성하였다. 메모리가 정상적으로 저장된 것을 확인할 수 있다. 메모리에서 1바이트를 읽어 r4에 저장하고, CMP명령어를 이용해서 r4과 r6을 비교하는 과정의 레지스터의 값 변경이 정상적으로 진행되었다. GT, MI, EQ flag의 조건을 만족하는 경우 해당하는 값으로 저장된 것을 확인할 수 있다. 어셈블리에서 조건부 실행을 연습할 수 있는 문제였다. CPSR의 각 비트의 의미와 flag를 이해하는 것이 필요했다.

(2) Exercise2

메모리에 값을 순서대로 넣어서 그대로 불러오면 역순으로 값이 저장되고, 메모리에 값을 거꾸로 넣어서 불러오면 원래의 순서대로 값이 저장된 것을 확인할 수 있다. LDR과 STR명령어를 이용해서 메모리에 값을 저장하거나 메모리의 값을 읽어오는 것을 연습할 수 있는 문제였다. Little-Endian 방식을 이용한다는 사실을 알 수 있었다. 또한 사용할 메모리의 주소 값을 지정할 때 사용한 Addressing mode에 대해 알아보고 응용할 수 있었다.

---

[i] Current Program Status Register, https://www.keil.com/pack/doc/cmsis/Core_A/html/group__CMSIS__CPSR.html#details

[ii] Little-Endian, https://www.techopedia.com/definition/12892/little-endian