어셈블리프로그램 설계 및 실습
과제1 보고서

담당교수	이형근 교수님
실습분반	화요일 (2분반)
소속	컴퓨터정보공학부
학번·성명	2023202027 정정윤
제출일	2024년 11월 12일

1. Problem Statement

본 과제는 다음을 목적으로 한다.

- ARM 어셈블리 기본 명령어의 이해와 활용
- 스택과 메모리 관리 능력 향상
- 재귀 함수의 구현 및 이해
- 레지스터와 메모리 조작
- 성능 분석 및 최적화 기법 학습

과제의 문제 1번은 Second Operand에서 Shift연산을 활용하여 10!을 계산하는 것으로, 의도는 MUL이 아닌 명령어를 통한 곱셈 연산 값 계산이다. 기본 명령어의 Operand로 무엇이 올 수 있는 지, 그리고 Second Operand에서 수행할 수 있는 Shift연산의 사용에 대해 잘 이해하고 있는 지 확인하고자 하는 과제로 볼 수 있다. 위 목적에서는 ARM 어셈블리 기본 명령어의 이해와 활용을 요구한다.

과제의 문제 2번은 스택과 재귀함수를 이용하여 10!을 계산하는 것이다. 서브루틴의 라벨이 지정되었으며, 이를 반복하여 수행하는 것으로 전제로 한다. 스택에 값을 넣고 꺼내오는 방법, 서브루틴의 처리 과정 속 스택의 처리 방법 등에 대해 이해하는 것을 목적으로 한다. 타 프로그래밍 언어에서 서브루틴을 처리할 시기에 스택을 어떻게 이용하는 지 어느정도 이해하고 있을 것을 필요로 한다. 위 목적에서는 기본 명령어의 이해와 활용, 스택과 메모리 관리 능력 향상과 재귀 함수의 구현 및 이해를 요구한다.

과제의 문제 3번은 3과 17의 곱을 각각 순서를 바꾸어 MUL 명령어로 수행하여 성능의 차이가 발생하는가에 대해 생각해보고, 이를 최적화하는 법에 대해서도 생각해보는 것을 목적으로 한다. ARM 어셈블리가 MUL 연산을 어떻게 수행하는 가에 대해 이해할 필요가 있다. 위 목적에서는 기본 명령어의 이해와 활용과 성능 분석 및 최적화 기법 학습을 요구한다.

과제의 문제 4번은 8개의 레지스터의 값을 각각 지정된 레지스터의 값과 교환하는 것으로, 여러 레지스터의 값을 한 번에 메모리/스택에 저장하고, 저장된 값을 하나씩 또는 동시에 여러 개 불러와 다시 레지스터에 넣는 것을 요구한다. 여러 값을 동시에 메모리에 저장하는 명령어인 STM과 불러오는 명령어인 LDM의 사용 요령을 요구한다. 위 목적에서는 기본 명령어의 이해와 활용, 스택과 메모리 관리 능력 향상, 레지스터와 메모리 조작을 요구한다.

2. Method

문제 1

단순히 10!로 생각하는 것이 아닌 N!로 생각하고 문제를 해결하였다. 이진수의 곱셉은 한쪽 수를 다른 수의 이진수 비트 위치마다 하나씩 하여 더하여준 것과 같다. 첫 번째 수에 대해 두 번 째 수의 각 비트별로 그 비트위치까지 첫 번째 수에 비트 쉬프트 연산을 취해준 후, 레지스터에 더하여 주는 방식으로 구현하였다. 이 확인을 위해 두 번 째 수가 그 비트에 1을 가지고 있는 지 확인할 필요가 있는데, 코드 상에서는 4번 비트부터 확인하도록 되어 있다. 1을 현재 비트 위치까지 비트 쉬프트 한 후, 두 번 째 수와 ANDS 명령을 수행해 NE Condition이 적합하다면 해당 비트 값은 1인 것이다. 따라서, 해당 비트까지 첫 번째 수를 왼쪽 비트 쉬프트 한 수를 최종 값을 저장하는 레지스터에 저장한다.
현재 비트가 -1이라면 곱연산이 끝난 것이다.

곱연산이 끝났다면 곱연산 최종 값을 현재 값이라 생각하고 곱연산의 두 번재 값을 하나 줄여 다시 곱연산을 수행한다. N이 0이 되었다면 시스템콜을 수행하며 종료한다.

문제 2

1번 문제와 목표는 같으나, MUL 연산을 통해, 그리고 스택과 재귀함수를 이용해 구현하는 것을 전제로 한다. 팩토리얼 연산 상 서브루틴으로 넘어갈 때 저장해두어야 하는 값은 N값밖에 없으므로, 스택 포인터에는 그 값을 저장하는 r0와 복귀 위치를 기록하는 lr만 push한다. 반환값은 r1을 통해 받을 수 있도록 MUL 연산이 종료되면, r1에 값을 저장한다. MUL 연산은 목표 레지스터와 대상 레지스터가 같을 수 없으므로, r2에 저장 후 r1으로 값을 다시 돌린 후 BX를 수행하도록 하였다.

따라서 재귀의 작동 순서는 CMP를 통해 현재 N이 1보다 작다면 r1에 1을 저장 후 BX, 아니라면 r0와 lr을 스택에 저장한 후, N-1을 r0에 두고 다시 이 루틴을 호출한다. BX를 통해 원래 위치로 돌아왔다면 다시 스택에서 r0와 lr값을 불러온다. 이후, 반환값인 r1과의 MUL을 수행, r2에 저장한 후, 그 값을 r1으로 돌려 반환값으로 사용한다.
모두 호출된 뒤에는 자동적으로 done으로 넘어가 메모리에 값을 저장하고 시스템콜을 하며 종료된다.

문제 3

이 문제는 로직을 구현하는 것이 문제가 아니기에 Method에는 단순히 비교하는 방법만 서술하였다. MUL 연산으로 3*17과 17*3을 각각 수행한다. 이후, Register 창에 있는 Internal의 States를 확인하면 연산이 얼마나 수행되었는 지 알 수 있다. ARM9E-S에서는 물론, 해당 연산에 있어 차이가 발생하지 않는다는 사실이 있지만, 이전 버전에서 차이가 나는 이유와 현 버전에서 차이가 나지 않는 이유에 대해서 다시 한 번 생각해보는 것으로 대체한다.

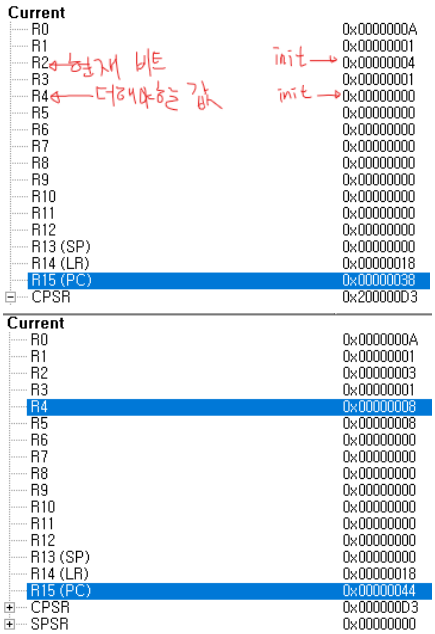
문제 4

레지스터에 저장된 값을 지정된 레지스터로 옮겨야 한다. 각각 r0은 r1으로, r1은 r6로 r2는 r0로 r3는 r2로 r4는 r7으로, r5~r7은 r3~r5로 옮겨야 한다. 한 번에 여러 값을 스택으로 저장하는 것을 요구하였기도 하고, 어차피 일단 다 저장해두지 않으면 교환 과정에서 레지스터의 값이 바뀌어 버릴 수 있기 때문에, STMDB로 스택에 r0~r7의 값을 PUSH해 두었다. 이후 r0의 값을 요구하는 레지스터부터 하나씩 다시 불러온다. 불러오는 것은 LDMIA로 수행할 수 있으며, 스택포인터는 아래에서부터 값을 저장하기에 저장은 내림차순으로 불러오기는 오름차순으로 주소를 변경하며 불러온다. R5~r7의 값을 r3~r5로 옮길 때는 한 번에 하나씩 불러오는 것보다 여러 개를 동시에 불러오는 것이 더 적은 명령을 수행하기에 한 번에 불러오도록 한다. 순서가 반대가 되지 않기에 여전히 LDMIA를 통해 수행한다.

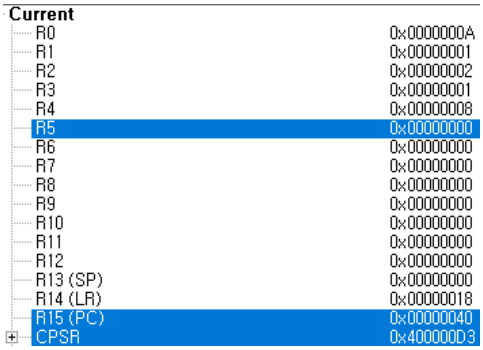
3. Result

문제 1

곱연산의 수행 과정 직전에는 r2에 현재 비트가 정상적으로 들어가고 r4비트가 더해야 할 값을 정상적으로 초기화한다. r1비트는 현재 값으로 10을 곱하기 이전 상태이므로, 정상적으로 1로 표기되어 있다.

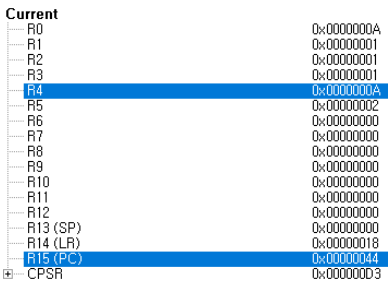


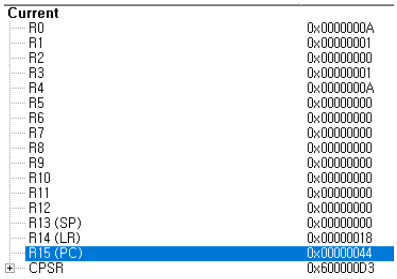
한 번 순환 후에는 r2가 가리키는 비트가 하나 줄어들고, A는 아래에서 3번 비트가 1이므로, 곱연산이 수행되어 r4에 r1을 왼쪽 쉬프트 3 한 값이 더해졌다. R5는 AND연산 한 값이 저장되는 곳으로, 3번 비트이므로, 현재 상황에서 3이 저장되어 있다.



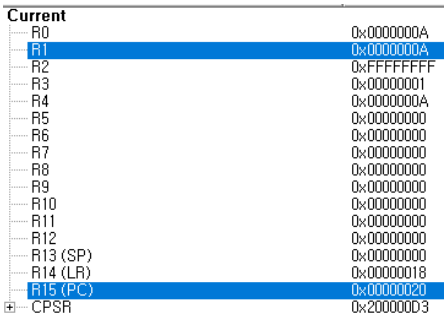
R2가 2번 비트를 가리키고 있는 상황에서는 A가 2번 비트에 0을 가지고 있으므로, R5값이 0이 되게 된다. 따라서, Z가 1이 되므로, r4에 비트 쉬프트 연산 값이 더해지지 않은 것을 볼 수 있다.

1번 비트를 가리킬 때에는 A가 해당 위치에 1을 가지므로, R5값이 AND연산한 결과인 2가 되고, r4에 비트 쉬프트 연산 결과가 들어가게 된다.

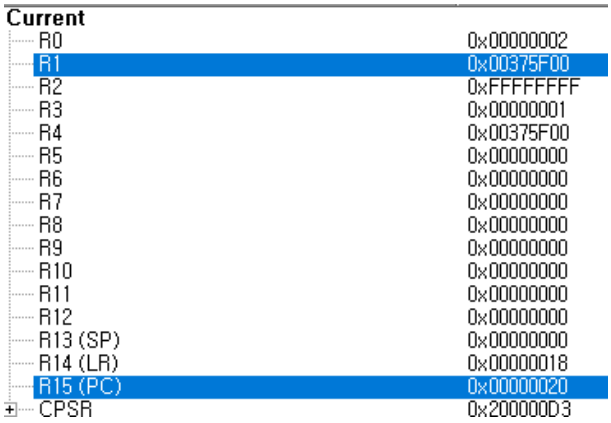




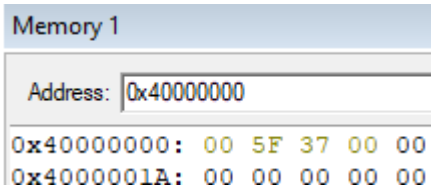
0번 비트를 가리키고 있을 때는, A가 해당 비트에 0 값을 가지므로, r4에 값이 더해지지 않는다.



R2가 -1이 되어서 루프가 종료되며, r1에 r4의 값이 저장되게 된다. 이후 r0를 하나 감소시킨 후 위 루틴을 반복하게 된다. 이를 통해 팩토리얼 연산을 수행한다.



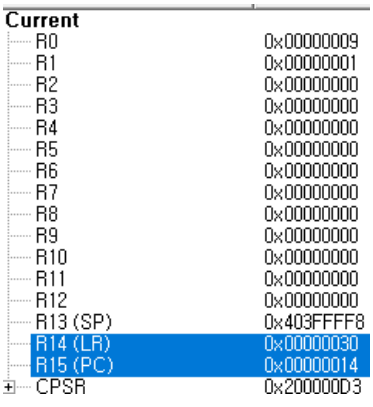
최종적으로 r1에 375F00값이 저장되며, 루틴이 종료된다. 이에 따라 done label로 이동하며, 메모리에 값이 저장되게 된다.

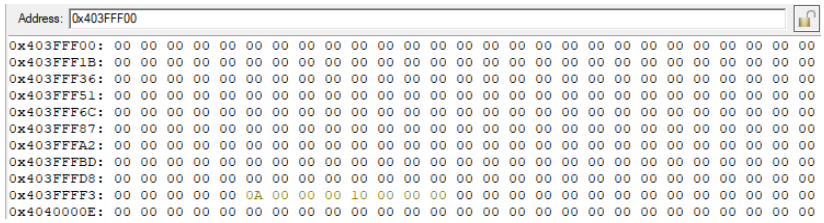


문제 2

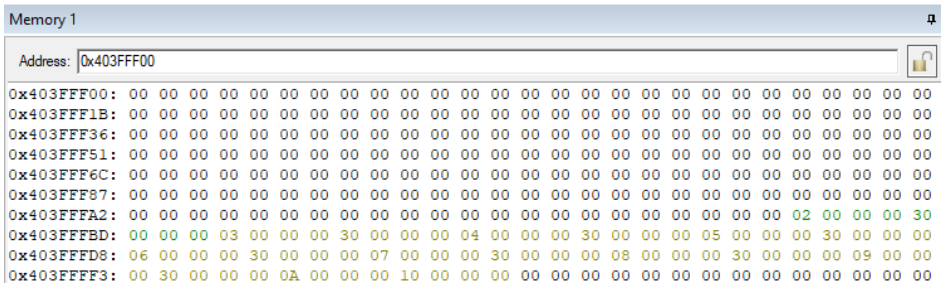
먼저 N값을 r0에 그리고 값을 곱하게 될 r1에 미리 1을 저장해둔다.

N=10인 경우에는 팩토리얼 재귀함수가 반환하는 값이 N=9인 경우를 요구하기 때문에, 스택포인터에 저장만 하고
이후 서브루틴으로 진입하게 된다. 매개변수 n으로 r0을 넘겨준 것과 같은 것으로 해석할 수 있다.

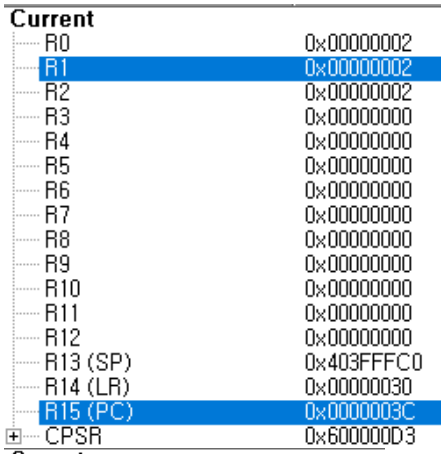




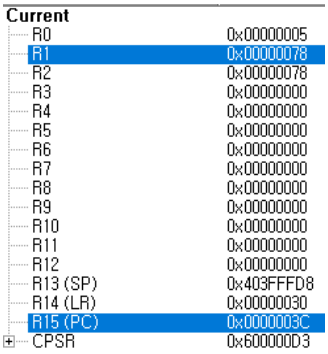
스택포인터가 가리키는 주소에 0A와 돌아갈 lr이 저장되었음을 확인할 수 있다.



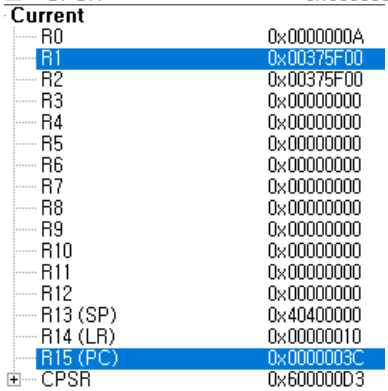
1까지 가서 1에서 r1을 반환한 이후로는 BX 호출을 하며, 스택포인터에서 다시 읽어오게 된다. 02까지 저장되고, 다시 호출되고 있음을 Memory 창에서 확인할 수 있다.



02가 다시 불러와져 r1과 곱하여 저장된 상태로 r1이 2가 되었음을 확인할 수 있다.

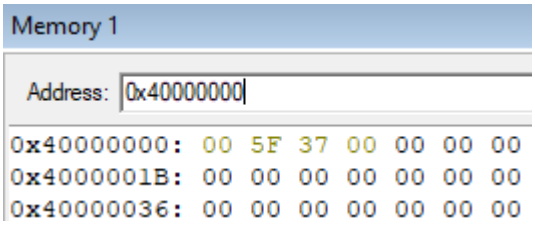


r0에 5가 다시 불러와진 시점에서, 레지스터 상태는 다음과 같다. 1부터 5까지 곱한 값이 들어간 상태라고 볼 수 있다.



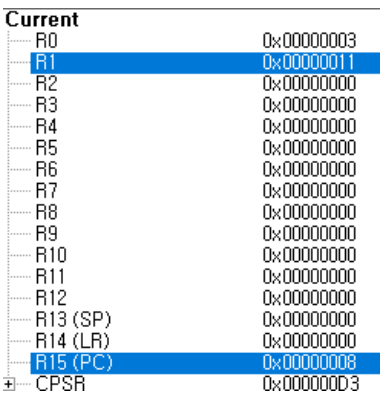
10값인 A를 불러올 시점에서는 lr이 10으로 설정되며, 처음 함수를 호출한 시점으로 돌아감을 알 수 있다.

이후에는 레지스터에 최종 연산값을 저장하며, 시스템콜을 하며 프로그램이 종료된다.

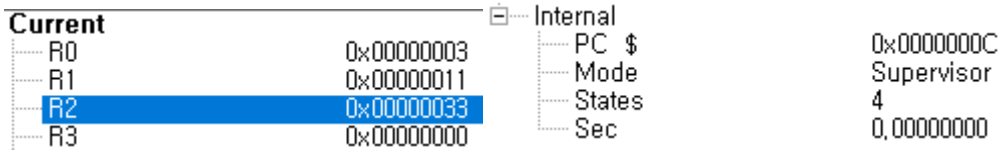


문제 3

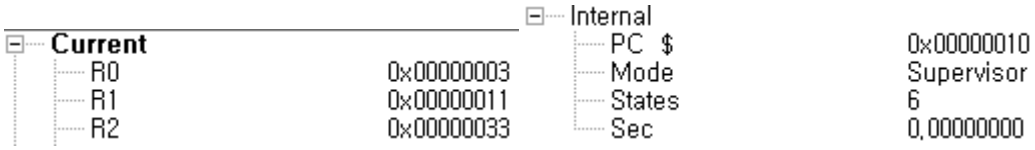
3과 17에 대한 곱연산의 수행 시간을 확인하는 것이기 때문에, 우선 r0와 r1에 3과 17을 넣는다.



현재 Internal 상황은 다음과 같다.



3 * 17로 첫 번째 Operand로 3, 두 번째 Operand로 17을 넣고 나온 결과이다. States가 2 증가하였음을 알 수 있다.



17 * 3으로 첫 번째 Operand로 17, 두 번째 Operand로 3을 넣고 나온 결과이다. States가 2 똑같이 2 증가하였음을 알 수 있다.

위 결과로는 3과 17 중 어느 것을 어느 Operand에 넣어도 걸리는 States 수가 같다는 사실을 알 수 있다.

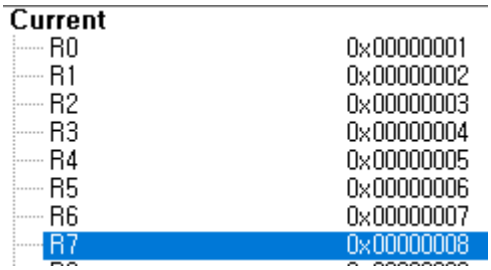
곱연산 수행 시에는 문제 1에서 사용하였던 것과 같이, 각 비트를 확인한 후 해당 비트만큼의 쉬프트 후 더할 필요가 있다. 따라서, 두 번째로 온 값에 대해 1의 개수가 많다면 변화가 생긴다는 가설을 제시할 수 있다. 1의 개수 자체가 많은 경우, 합연산 또한 그만큼 많이 수행해야 하기 때문이다.

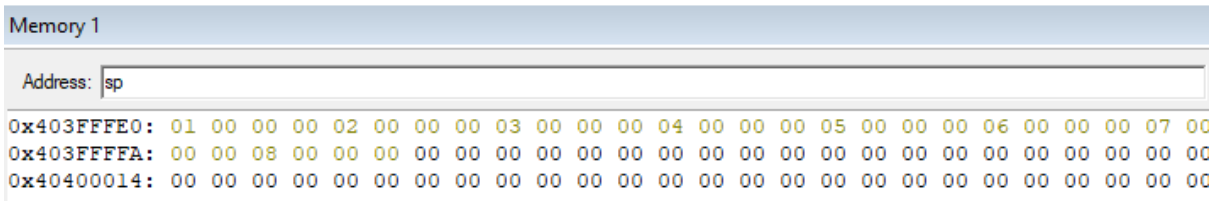
두 번째로 ARM 프로세서가 과거 2비트씩 확인하는 것과 버전이 올라감에 따라 더 많은 비트를 확인하는 점에서 해당 비트 수 단위에서 피연산자인 수 둘이 서로 차이가 나는 경우, (3과 17인 경우에는 2와 5이기 때문에 2단계 차이가 난다.) 연산 횟수에서 차이가 날 수 있다고 볼 수 있다.

현재 차이가 나지 않는 것에 대해서는, ARM 프로세서에서 곱연산 수행에 들어가는 Peripheral이 발전했을 가능성과 Keil 시뮬레이터가 MUL연산으로 인한 States 변화를 제대로 지원하지 않는다는 두 가정을 제시할 수 있다.

문제 4

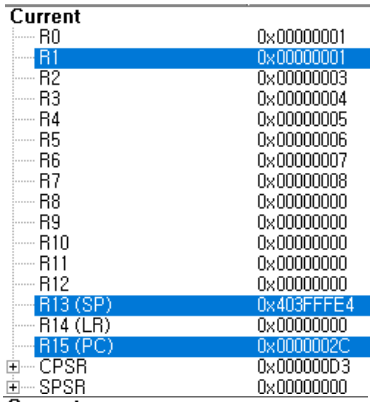
R0부터 R7까지 1부터 8까지 값을 초기값으로 넣고, 스택포인터도 권한이 있는 메모리 값으로 설정한다.





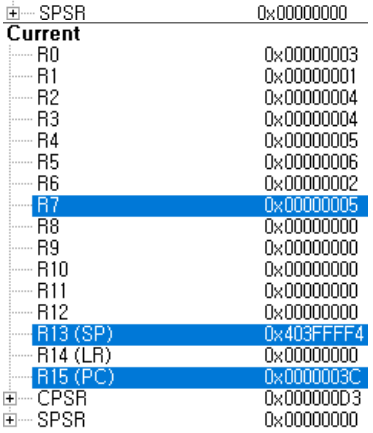
이후 STMDB 명령을 통해 스택에 값을 넣은 상태이다.

스택 포인터에 값을 넣으며, !를 통해 저장 이후의 위치까지 포인터를 움직여두었기 때문에, Memory에서 sp로 위치를 찾게 되면, 위와 같이 저장된 위치부터 메모리가 표시된다.

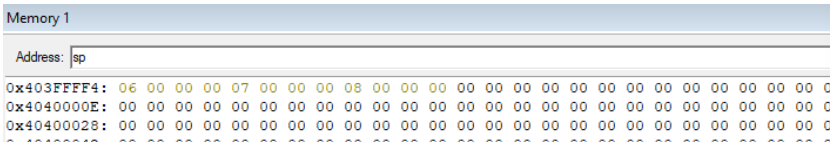


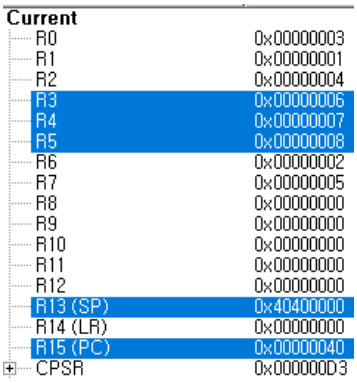
r0의 값은 r1으로 이동하도록 되어 있으므로, 스택의 맨 위부터 r0의 값을 저장하고 있으므로, LDMIA 명령 수행에 따라 r0 값이 들어와야 하며, 정상적으로 작동되고 있음을 알 수 있다.

sp!를 통해 sp값 또한 변경되어야 하며, 기존 0x403FFFE0에서 4가 증가한 0x403FFFE4가 되어있음을 확인할 수 있다. 스택포인터는 주소 위부터 기록하도록 되어있기 때문에 정상 작동하는 것으로 판단할 수 있다.



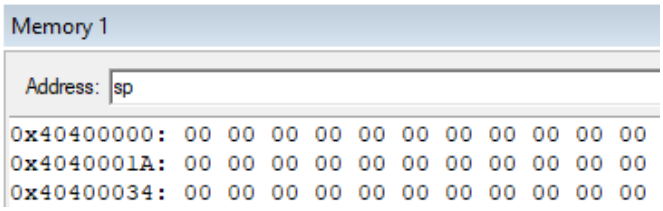
순서가 섞여있는 레지스터에 대해 모든 호출을 완료한 상태이다. R4에 있던 데이터까지 해당 방식의 작업을 요하였다. Memory에서도 5까지의 값이 스택에서 나갔음을 확인할 수 있다.





R3-R5는 R5-R7의 값을 순서대로 요구하고 있기 때문에 LDM명령으로 한 번에 스택에서 POP하도록 프로그래밍하였다. 결과적으로 명령줄 하나 (pc+4)와 함께 R3-R5의 값이 지정되 레지스터 값으로 변경되었다.

스택에서는 모든 값이 빠져나가 아무런 값도 저장되어있지 않음을 확인할 수 있다.



4. Discussion and Conclusion

uVision 프로그램의 사용법에 대해서 조금 더 파악하였으며, Debugging 화면 전반적인 내용과, 해당 화면에서 Register이외의 정보들에 대해서 어떤 방식으로 파악할 수 있는 지 알 수 있게 되었다. Memory에서 값을 읽어온 기록이나, Disassembly가 가리키고 있는 곳과 정확한 명령의 주소 정보 등을 파악하는 방법에 대해서 더 알게 되었다.

문제 1과 문제 2 수행 과정에 있어 B, BL, BX를 사용하지 않고 MOV를 사용하게 될 경우에 pc+8이 불러와지는 문제에 대하여, pipeline에 의해 생기는 현상임을 알게 되며, 이 해결을 위해 전반적인 Branch과정은 모두 지정된 명령어를 이용하는 것이 좋다는 사실에 대해 알고 프로젝트 프로그래밍에 있어 해당 과정을 적극적으로 이용하였다. 결과적으로 스택 저장은 lr을 포함한 연산값 저장용 레지스터의 값을 남겨두는 것에 사용하는 쪽이 Branch에서 pc나 lr 값을 남겨두는 것보다 더 적합하다는 사실을 알 수 있었다.

문제 3의 해결에 있어서는 States의 변화량이 달라지지 않았기에 더 명확한 원인 파악을 위해 Peripheral 단위로 확인할 수 있는 기능의 사용법에 대해 더욱 파악하고자 하고 있다.

Acknowledgement

(Optional) 다른 학생으로부터 도움을 받은 경우, 어떤 도움을 받았는 지 명시합니다.

Reference

어셈블리프로그램설계및실습 강의자료

ARM 공식 문서

명예서약서 (Honor code)

본인은 [어셈블리프로그램설계및실습] 수강에 있어 다음의 명예서약을 준수할 것을 서약합니다.

1. 기본 원칙

- 본인은 공학도로서 개인의 품위와 전문성을 지키며 행동하겠습니다.
- 본인은 안전, 건강, 공평성을 수호하는 책임감 있는 공학도가 되겠습니다.
- 본인은 우리대학의 명예를 지키고 신뢰받는 구성원이 되겠습니다.

2. 학업 윤리

- 모든 과제와 시험에서 정직하고 성실한 태도로 임하겠습니다.
- 타인의 지적 재산권을 존중하고, 표절을 하지 않겠습니다.
- 모든 과제는 처음부터 직접 작성하며, 타인의 결과물을 무단으로 사용하지 않겠습니다.
- 과제 수행 중 받은 도움이나 협력 사항을 보고서에 명시적으로 기재하겠습니다.

3. 서약 이행

본인은 위 명예서약의 모든 내용을 이해하였으며, 이를 성실히 이행할 것을 서약합니다.

학번	2023202027	**이름**	정정윤	**서명**	정정윤