어셈블리프로그래밍설계 및 실습 보고서

실험제목: Control flow & Data processing

실험일자: 2023년 09월 26일 (화)

제출일자: 2023년 09월 27일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 화 6,7 목 5

학 번: 2022202065

성 명: 박나림

1. 제목 및 목적 (3%)

A. 제목

Control flow & Data processing

B. 목적

조건부 명령문들에 대해 이해하고 적절한 사용법을 익히도록 한다. LDR, STR와 같이 메모리에 데이터를 저장하고 불러오는 방법을 공부하여 코드를 설계할 수 있도록 한다. 이와 관련하여 index method와 현재 사용되는 little-endian 방식을 이해하여 어셈블리 프로그래밍 능력을 습득할 수 있도록 한다.

2. 설계 (Design) (50%)

A. Pseudo code

1)

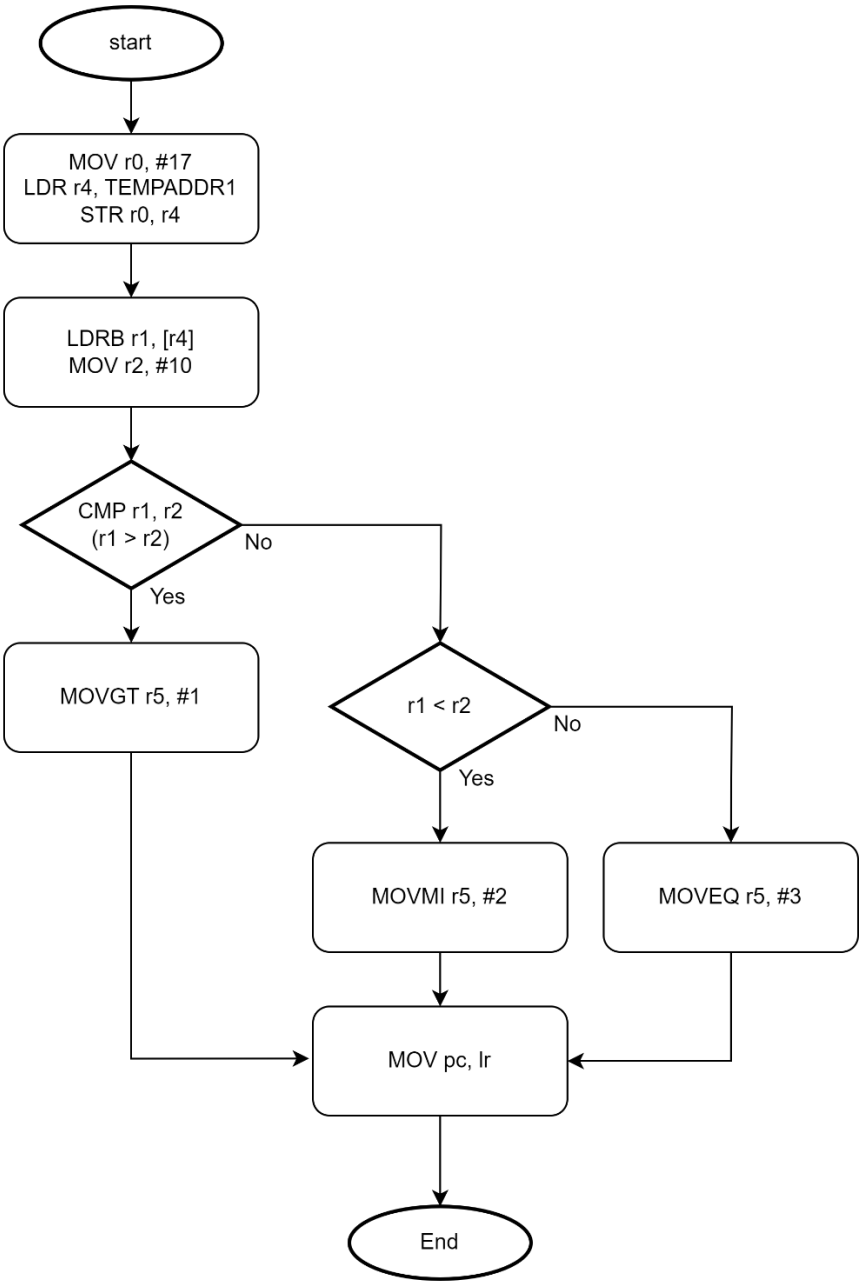
```
  AREA ARMex, CODE, READONLY
        ENTRY
start
        MOV r0, #17          ; r0에 0x11 저장
        LDR r4, TEMPADDR1 ; r4에 TEMPADDR1 주소 불러옴
        STR r0, [r4]         ; r4 주소에 r0의 값을 저장

        LDRB r1, [r4]        ; r4 주소에 저장된 값을 r1에 1byte 단위로 불러옴
        MOV r2, #10          ; r2에 0x0A 저장

        CMP r1, r2           ; r1-r2
        MOVGT r5, #1         ; r1이 크면 r5에 1 저장
        MOVMI r5, #2         ; r1이 작으면 r5에 2 저장
        MOVEQ r5, #3         ; 같으면 r5에 3 저장

TEMPADDR1 & &00001000   ;TEMPADDR1의 주소

        MOV pc, lr           ; lr의 값을 pc에 저장
        END
```

문제 2)

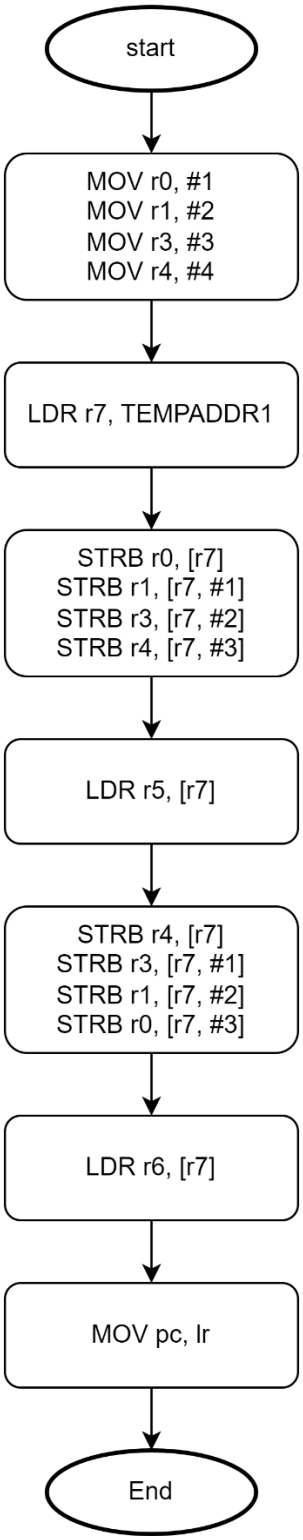
```
  AREA ARMex, CODE, READONLY
        ENTRY
start
        MOV r0, #1              ; r0에 1 저장
        MOV r1, #2              ; r0에 2 저장
        MOV r3, #3              ; r3에 3 저장
        MOV r4, #4              ; r4에 4 저장

        LDR r7, TEMPADDR1   ; r7에 TEMPADDR1 주소 불러옴
        STRB r0, [r7]           ; r7 주소에 r0의 값을 1byte단위로 저장
        STRB r1, [r7, #1]       ; Preindex 방식으로 offset으로 1byte 더한 주소에 r1 저장
        STRB r3, [r7, #2]       ; 2byte 더한 주소에 r3 저장
        STRB r4, [r7, #3]       ; 3byte 더한 주소에 r4 저장
        LDR r5, [r7]            ; 기존 r7 주소에 저장된 값을 r5에 불러옴

        STRB r4, [r7]           ; 위와 같은 방식에서 반대 순서로 저장
        STRB r3, [r7, #1]
        STRB r1, [r7, #2]
        STRB r0, [r7, #3]
        LDR r6, [r7]            ; r7 주소에 저장된 값을 r6에 불러옴

TEMPADDR1 & &00001000   ; TEMPADDR1의 주소

        MOV pc, lr              ; lr의 값을 pc에 저장
        END
```

B. Flow chart 작성

1)



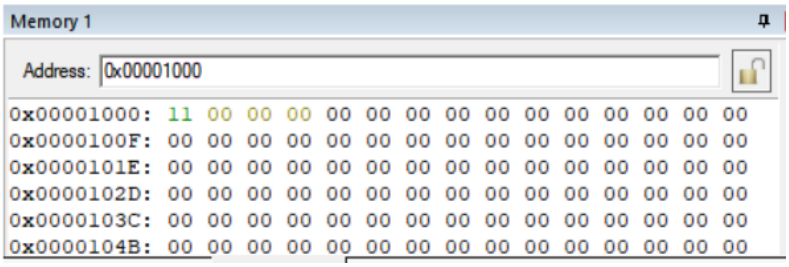
2)


start
MOV r0, #1
MOV r1, #2
MOV r3, #3
MOV r4, #4
LDR r7, TEMPADDR1
STRB r0, [r7]
STRB r1, [r7, #1]
STRB r3, [r7, #2]
STRB r4, [r7, #3]
LDR r5, [r7]
STRB r4, [r7]
STRB r3, [r7, #1]
STRB r1, [r7, #2]
STRB r0, [r7, #3]
LDR r6, [r7]
MOV pc, lr
End

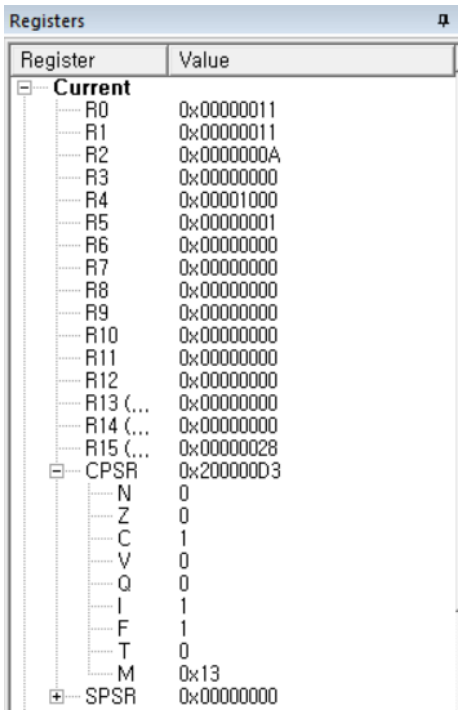
C. Result

1)

먼저 메모리에 임의의 값인 0x11을 저장한 모습이다.

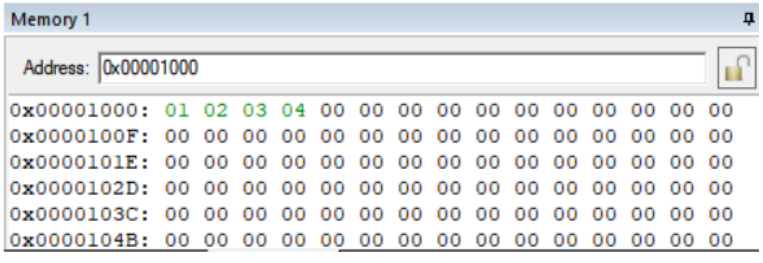


그 이후 r1에 해당 메모리 주소에 저장된 값을 불러온 뒤, r2에 0x0A를 저장하여 둘의 값을 비교한다. 이때 carry가 발생하여 C의 값이 1이 되고 r1의 값이 더 크므로 r5에 1이 저장되게 된다.

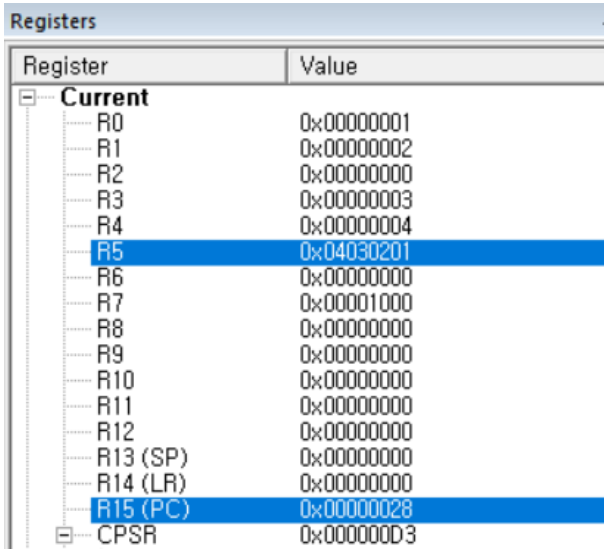


2)

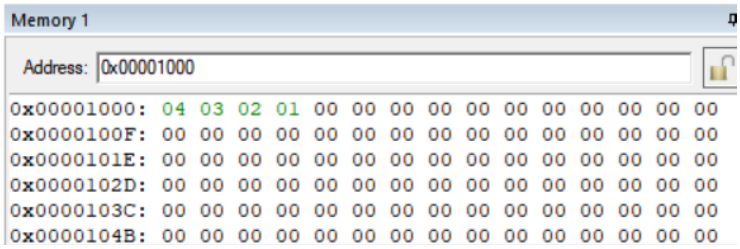
4개의 레지스터에 각각 값을 저장한 뒤, 메모리에 차례대로 값을 저장한 모습이다.

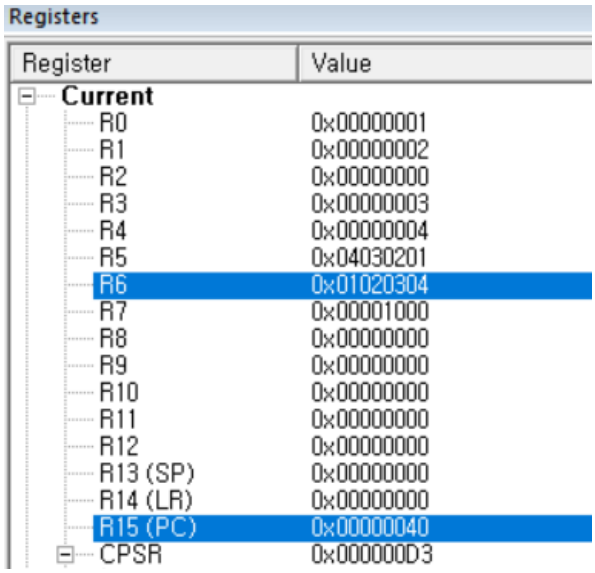


그리고 메모리로부터 4byte단위로 읽어와서 r5에 저장한다. 이때 little-endian 방식임으로 역순으로 저장된 걸 볼 수 있다.



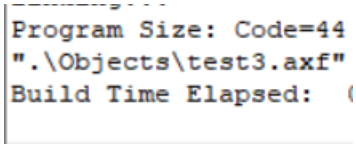
다음으로 r6에는 순서대로 저장하기 위해서, 다시 메모리에 반대 순서로 레지스터의 값들을 저장한다. 이렇게 하면 LDR할 때 역순으로 불러와지므로 원하는 결과값이 나오게 된다.



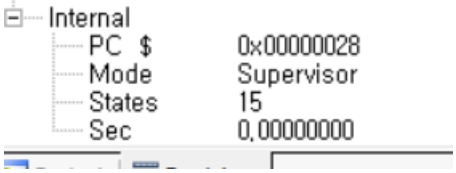


D. Performance

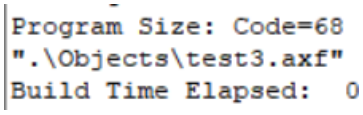
1-1) Code size: 44 byte



1-2) state: 15



2-1) Code size: 68 byte



2-2) state: 30

3. 고찰 및 결론

A. 고찰 (35%)

2번 문제를 진행할 때, 메모리에 레지스터의 값들을 저장하는 과정에서 postindex 방식을 사용하였다가 주소가 업데이트 되어서 r5에 해당 주소의 값을 불러올 때 빈 값을 불러오는 오류가 발생했었다. 이에 preindex로 바꾸어서 주소를 업데이트 시키지 않고 진행하였더니 정상적으로 결과 값이 나올 수 있었다.

또한 현재 little-endian 방식임으로 메모리에서 레지스터에 값을 불러올 때 역순으로 불러와지는데, 이에 r5는 원하는 순서로 저장이 됐으나 r6는 어떻게 순서대로 저장해야 될 지 잘 몰랐었다. 이에 다른 명령어들을 사용해야 하는지 고민하다, 메모리에 다시 반대로 저장하면 역순으로 불러올 때 순서대로 저장된다는 것을 깨달았다.

B. 결론 (10%)

postindex와 preindex의 차이점과 상황에 따라 어떤 것이 더 알맞은지 공부해볼 수 있었다. 이번 문제 같은 경우에는 원래 주소를 다시 불러와야 되기 때문에 값을 업데이트 하지 않는 preindex를 사용하는 게 적절한 선택이었던 것 같다.

또한 LDR과 STR을 사용하면서 메모리와 레지스터의 데이터 이동을 자세히 알 수 있었다. 그 중에서 LDR과 LDRB도 다르다는 점을 깨달았는데, LDR이나 STR처럼 쓰면 기본 4byte단위로 불러오고 LDRB나 STRB로 쓰면 1byte단위로 적용된다. 메모리에 값이 나타날 때는 두 자리 씩 끊어서 나타나지는데, 이는 1byte를 나타낸다는 점도 알 수 있었다.

조건부 실행 문제에 관해서는 CMP(r1-r2), GT(>), LT(<), GE(>=), LE(<=), MI(<), EQ(==) 등 다양한 문법을 적용해보면서 더 잘 익힐 수 있었다.

4. 참고문헌 (2%)

이준환 / 어셈블리프로그램설계및실습 강의자료 / 광운대학교(컴퓨터공학과) / 2023