어셈블리프로그래밍설계 및 실습 보고서

실험제목: control flow & data processing

실험일자: 2023년 10월 10일 (화)

제출일자: 2023년 10월 12일 (목)

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

조건에 따라 Branch 명령어로 이동할 수 있도록 코드를 설계해본다. 메모리를 할당하여 문자열 등의 데이터를 저장하고 사용할 수 있게 한다. 이러한 코드를 만들 때 performance를 생각하며 효율적으로 짤 수 있도록 한다.

2. 설계 (Design) (50%)

A. Pseudo code

Problem 1)

```
AREA ARMex, CODE, READONLY
        ENTRY
start
        MOV r0, #3   ;a
        MOV r1, #1   ;b
        MOV r3, #3   ;c
        MOV r4, #4   ;d
        MOV r5, #2   ;2
        B Example_1   ;Branch 명령어: Example_1 함수로 이동


Example_1   ;함수 시작
        MUL r6, r4, r5   ;r6: e, e=d*2
        ADD r6, r6, r3   ;e=c+(d*2)
        SUB r0, r0, r1   ;a=a-b
        ADD r6, r6, r0   ;e=(a-b)+(c+(d*2))

        LDR r8, TEMPADDR1   ;저장할 메모리 주소 불러옴
        STR r6, [r8]   ;해당 메모리에 결과 저장


TEMPADDR1 & &40000   ;메모리 주소

        MOV pc, lr   ;함수 반환
        END
```

Problem 2)

```
AREA ARMex, CODE, READONLY
        ENTRY
start
        MOV r0, #0   ;문자열의 index를 count 하는 용
        LDR r1, C_string_m   ;문자열의 주소 불러옴

        B Strlen   ;strlen 함수로 이동


Strlen   ;strlen 함수 시작
        LDRB r2, [r1, r0]   ;문자열의 주소를 한 칸 씩 업데이트 하면서 해당 문자를 r2에 불러옴
        CMP r2, #0   ;해당 문자가 문자열의 끝인 null 값인지 비교
        BEQ ENDSTR   ;문자열 끝이 맞다면 종료 branch로 이동

        ADD r0, r0, #1   ;index 한 칸 이동
        B Strlen   ;다시 함수 반복


ENDSTR   ;문자열 길이 반환 부분
        LDR r8, TEMPADDR1   ;저장할 메모리 주소 불러옴
        STR r0, [r8]   ;해당 메모리에 결과 저장

        MOV pc, lr   ;함수 반환

TEMPADDR1 & &40000   ;메모리 주소

C_string_m DCD C_string   ;문자열을 저장하고 있는 주소, DCD로 메모리 할당 및 초기화
C_string DCB "Hello",0   ;문자열을 DCB로 1byte단위로 메모리 할당 및 해당 값으로 초기화

        END
```

Problem 3-1) loop 이용 덧셈

```
AREA ARMex, CODE, READONLY
```

```
        ENTRY
start
        MOV r0, #1   ;덧셈 시작 값
        MOV r1, #0   ;덧셈 결과 저장
        B LoopADD   ;반복문으로 이동


LoopADD   ;반복문 시작
        ADD r1, r1, r0   ;서로 더함
        ADD r0, r0, #1   ;r0값 1 증가
        CMP r0, #11   ;r0이 11이면 아래로 내려가서 종료
        BNE LoopADD   ;11이 아니면 다시 반복

        LDR r2, TEMPADDR1   ;저장할 메모리 주소 불러옴
        STR r1, [r2]   ;해당 메모리에 결과 저장


TEMPADDR1 & &40000   ;메모리 주소

        END
```

Problem 3-2) n(n+1)/2 공식 이용 덧셈

```
AREA ARMex, CODE, READONLY
        ENTRY
start
        MOV r0, #10   ;덧셈 마지막 값
        MOV r1, #0   ;덧셈 결과 저장

        MUL r1, r0, r0   ;n^2
        ADD r1, r1, r0   ;n^2 + n
        ASR r1, r1, #1   ;Arithmetic Shift Right, 오른쪽으로 한 칸 시프트 하여 2로 나누게 됨

        LDR r2, TEMPADDR1   ;저장할 메모리 주소 불러옴
        STR r1, [r2]   ;해당 메모리에 결과 저장


TEMPADDR1 & &40000   ;메모리 주소
```

```
        MOV pc, lr   ;함수 반환
        END
```

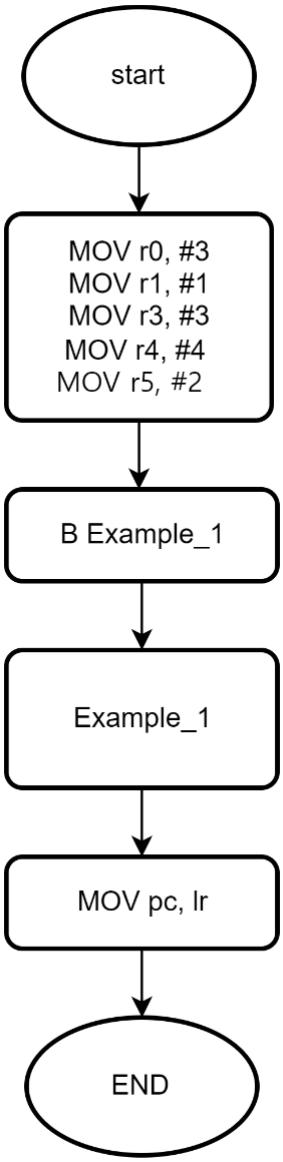
Problem 3-3) Unlooping 이용 덧셈

```
AREA ARMex, CODE, READONLY
        ENTRY
start
        MOV r0, #1   ;덧셈 시작 값
        ;r2를 1씩 증가 시키면서 차례대로 덧셈
        MOV r1, #2
        ADD r0, r0, r1

        MOV r1, #3
        ADD r0, r0, r1

        MOV r1, #4
        ADD r0, r0, r1

        MOV r1, #5
        ADD r0, r0, r1

        MOV r1, #6
        ADD r0, r0, r1

        MOV r1, #7
        ADD r0, r0, r1

        MOV r1, #8
        ADD r0, r0, r1

        MOV r1, #9
        ADD r0, r0, r1

        MOV r1, #10
        ADD r0, r0, r1
```

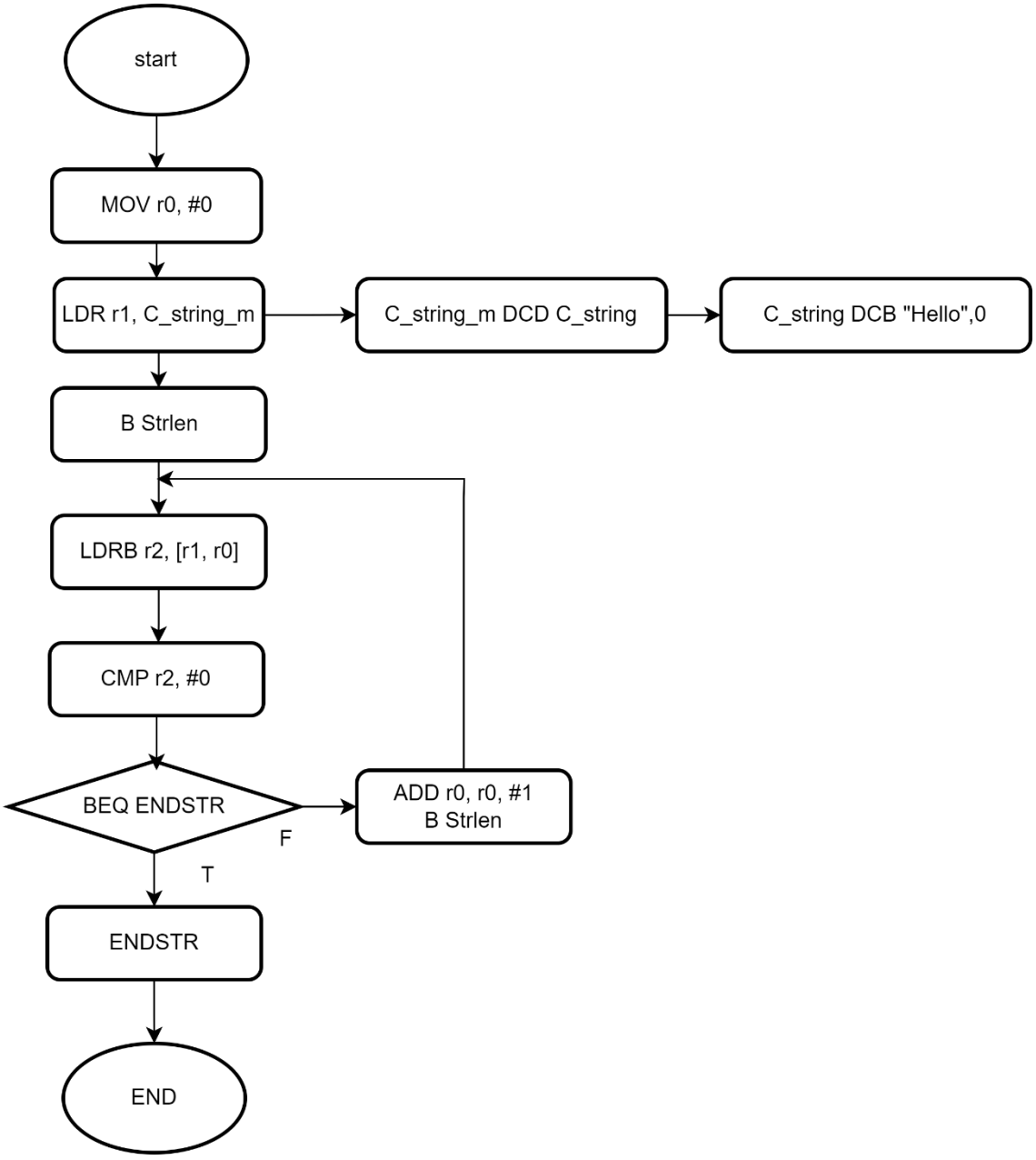
```
        LDR r2, TEMPADDR1   ;저장할 메모리 주소 불러옴
        STR r0, [r2]   ;해당 메모리에 결과 저장


TEMPADDR1 & &40000   ;메모리 주소

        MOV pc, lr   ;함수 반환
        END
```

B. Flow chart 작성

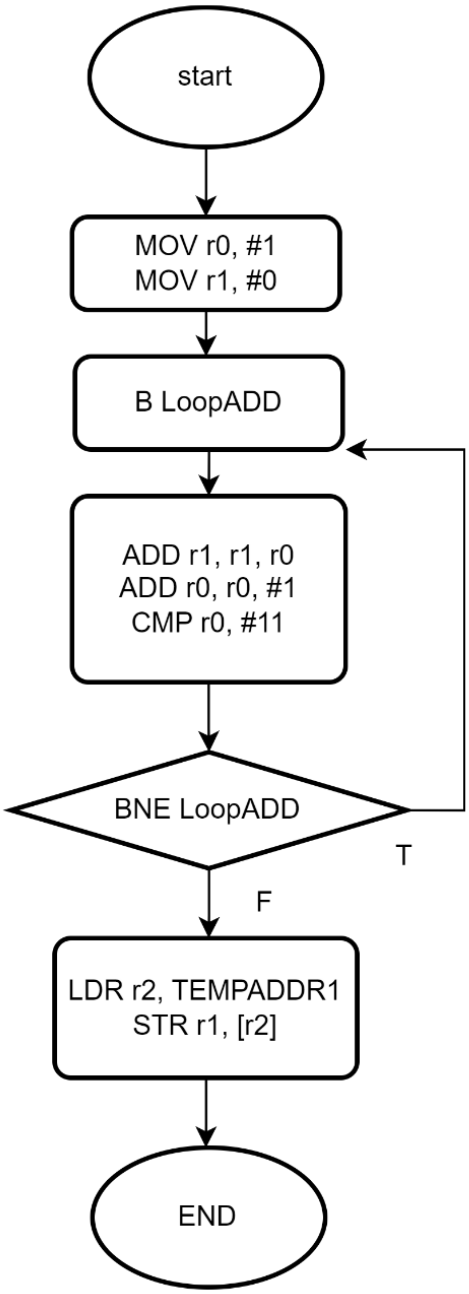
Problem 1)



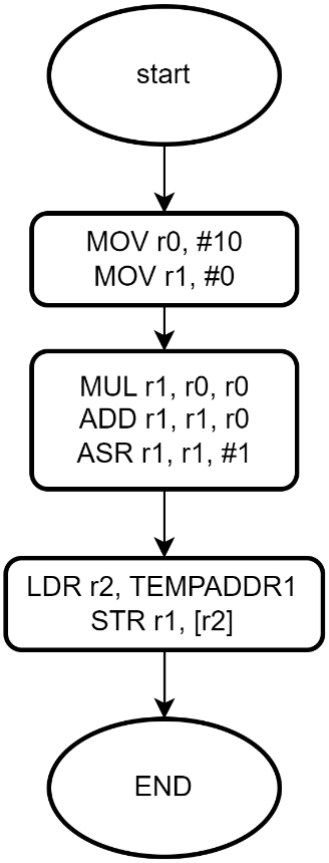
Problem 2)


start
MOV r0, #0
LDR r1, C_string_m
C_string_m DCD C_string
C_string DCB "Hello",0
B Strlen
LDRB r2, [r1, r0]
CMP r2, #0
BEQ ENDSTR
F
ADD r0, r0, #1
B Strlen
T
ENDSTR
END

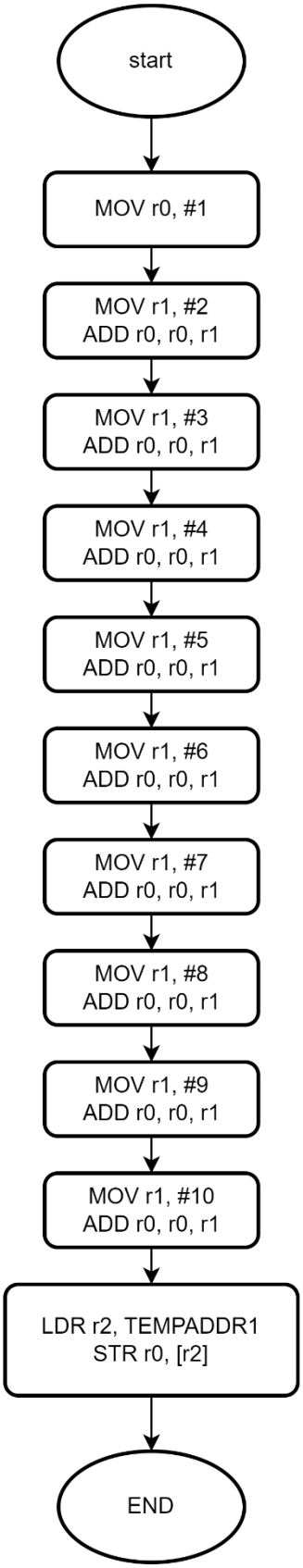
Problem 3-1)



Problem 3-2)



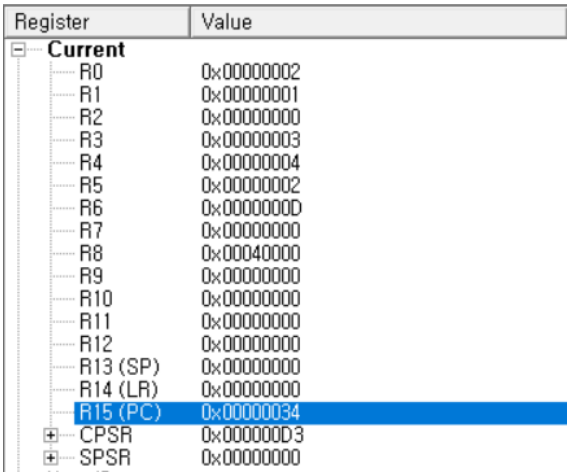
Problem 3-3)

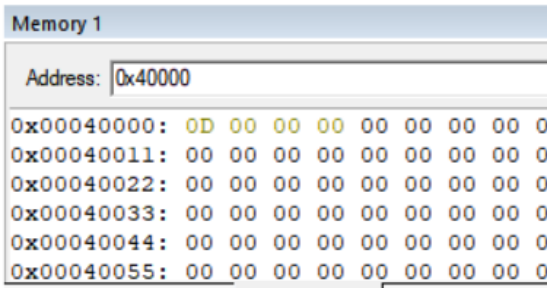

start
MOV r0, #1
MOV r1, #2
ADD r0, r0, r1
MOV r1, #3
ADD r0, r0, r1
MOV r1, #4
ADD r0, r0, r1
MOV r1, #5
ADD r0, r0, r1
MOV r1, #6
ADD r0, r0, r1
MOV r1, #7
ADD r0, r0, r1
MOV r1, #8
ADD r0, r0, r1
MOV r1, #9
ADD r0, r0, r1
MOV r1, #10
ADD r0, r0, r1
LDR r2, TEMPADDR1
STR r0, [r2]
END

C. Result

Problem 1)

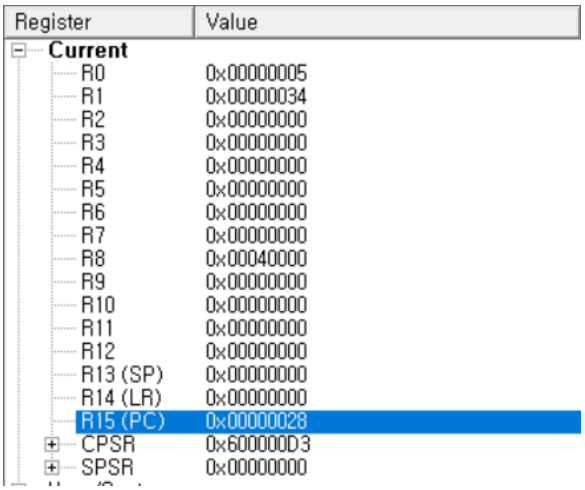
임의의 수 a, b, c, d를 r0~r4 레지스터에 각각 값을 할당한 뒤, r5에 2를 넣고 함수로 보내어 문제대로 식을 계산한다. 그 결과를 r6에 넣어서 메모리의 주소를 가지고 있는 r8을 이용하여 해당 메모리에 결과 값인 D를 넣은 모습이다.

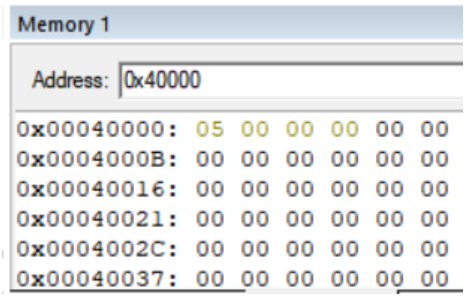




Problem 2)

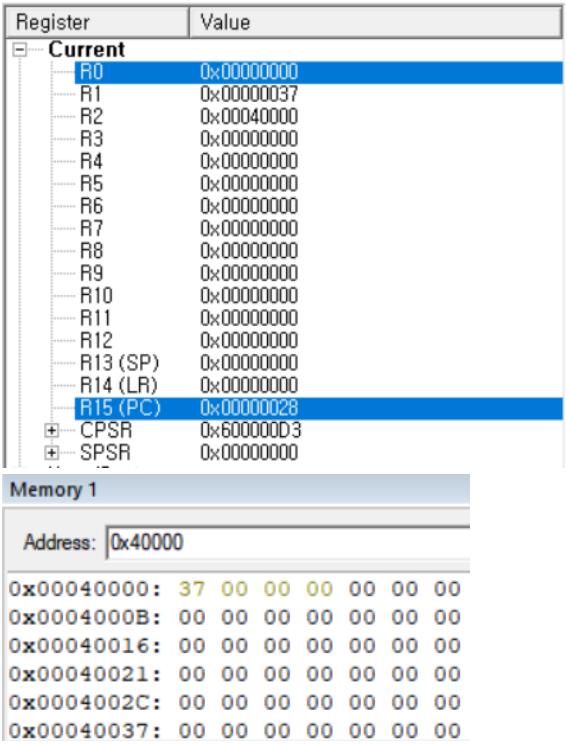
Strlen 함수로 "Hello" 문자열을 넣어 문자열의 길이를 구한 모습으로, r0은 그 개수를 저장하여 메모리 주소를 가지고 있는 r8을 이용하여 메모리에 최종 결과인 5를 저장한다. R1은 문자열의 해당 index에 있는 문자의 정보로 업데이트 된다.





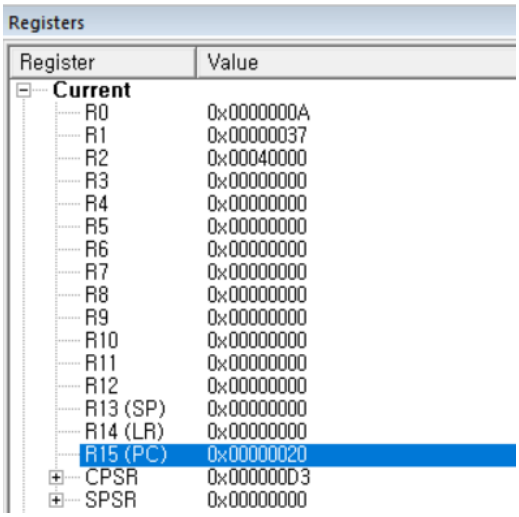
Problem 3-1)

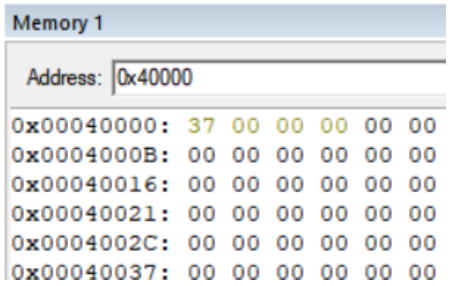
R0를 덧셈 시작 값부터 시작하여 1씩 증가 시킨다. 이러한 r0값을 이용하여 덧셈한 결과를 r1에 저장한다. 이후 최종 결과인 37을 메모리에 저장한 모습이다.



Problem 3-2)

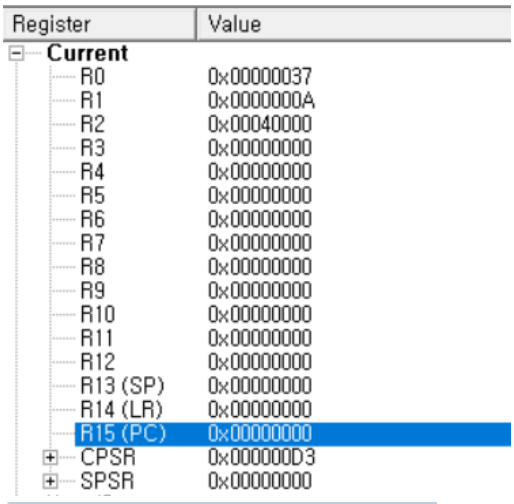
덧셈의 마지막 값인 10을 r0에 저장하고, 공식을 이용하여 계산한 결과를 r1에 저장한다. 이후 최종 결과인 37을 메모리에 저장한 모습이다.

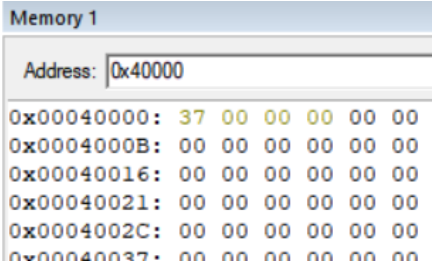




Problem 3-3)

R0을 덧셈 시작 값인 1부터 시작하여 차례로 증가시키며 r0와 더한다. R0엔 최종 덧셈 결과인 37이 저장되며, 이를 메모리에 저장한 모습이다.





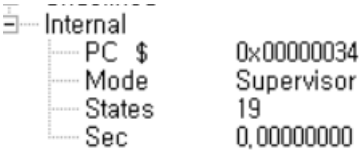
D. Performance

1)

-Code size: 56

```
Program Size: Code=56
".\Objects\test3.axf"
Build Time Elapsed:  0
```

-State: 19

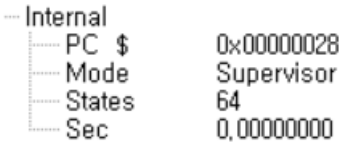


2)

-code size: 60

```
Program Size: Code=60
".\Objects\test3.axf"
Build Time Elapsed:  0
```

-state: 64

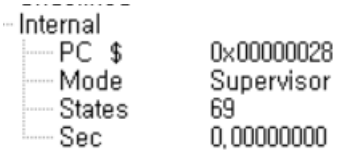


3-1)

-code size: 40

```
Program Size: Code=40
".\Objects\test3.axf"
Build Time Elapsed:  0
```

-state: 69

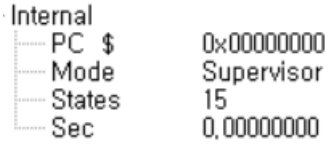


3-2)

-code size: 36

```
Program Size: Code=36
".\Objects\test3.axf"
Build Time Elapsed:
```

-state: 15

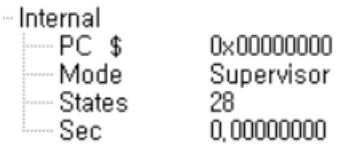


3-3)

-code size: 92

```
Program Size: Code=92
".\Objects\test3.axf"
Build Time Elapsed:
```

-state: 28



3. 고찰 및 결론

A. 고찰 (35%)

문제 2에 대한 코드를 작성할 때, strlen을 구현하기 위해서는 먼저 임의의 문자열을 받아와야 하는데 그 부분에서 여러 시행착오를 겪었다. 처음에는 단순히 'C_string DCB "Hello",0'만 쓰고 LDR을 통해 문자열의 주소를 불러와서 사용하였다. 이렇게 하면 strlen 함수에서 CMP를 통해 해당 문자 값을 비교할 때 문자의 값이 아니라 주소의 값과 비교하게 되어 절대 null인 0이 나올 수가 없어서 무한 반복하게 되는 상황이 발생하였다. 이에 문자열의 주소를 한 칸 씩 업데이트 하면서 해당 문자를 직접 비교할 수 있도록 'LDRB r2, [r1, r0]' 부분을 함수에 추가하였다. 그러나 이렇게만 하면 r2 레지스터의 값이 처음부터 0으로 나오게 되면서 바로 함수가 종료되었다. 문자열의 주소에 문자가 제대로

저장되지 않아서 발생한 문제인 것 같다는 생각이 들어서, 문자열 뿐만 아니라 문자열을 저장하고 있는 주소까지 메모리에 할당 및 초기화 하는 'C_string_m DCD C_string'까지 추가하였다. 이렇게 하니 함수가 정상적으로 작동될 수 있었다.

B. 결론 (10%)

이번 문제를 통해 함수를 branch문으로 구현할 수 있다는 것을 잘 알게 되었다. 또한 문자열에 관한 코드를 다루면서 DCD와 DCB에 대한 명령어도 새롭게 쓸 수 있었다. 이 명령어는 각각 4byte, 1byte로 메모리를 할당 및 초기화 시킨다. 또한 DCI로 하면 4byte 단위로 DCD와 비슷하게 작동되지만 비트패턴까지 넣어줄 수 있다. 비트패턴 형식으로 메모리에 할당 및 초기화 할 때 사용하면 될 것 같다.

또한 문제 3-2번을 풀면서, ASR에 대해 알게 되었다. 나눗셈을 할 수 있는 명령어인데, 오른쪽으로 해당하는 값만큼 시프트 연산하여 값을 구할 수 있었다. 이번 실습에서는 2로 나눠야 해서 1칸 시프트 하였는데, 예를 들어 10을 2로 나눌 때 1010을 오른쪽으로 1칸 시프트 하면 0101인 5가 되므로 정상적인 결과 값이 나오게 된다. 다른 나눗셈을 할 때에도 특정 값이 나오도록 시프트 연산을 사용하면 결과를 구할 수 있을 것 같다.

4. 참고문헌 (2%)

이준환 / 어셈블리프로그램설계및실습 강의자료 / 광운대학교(컴퓨터공학과) / 2023