# 어셈블리프로그램설계및실습 보고서

## 과제 주차: 4주차


학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화 6,7 목 5

학 번: 2019202021

성 명: 정성엽


제 출 일: 2022.10.17(월)

## 1. Problem Statement

Multiplication operation을 사용하는 것과 Second operand를 사용한 코드의 성능 차이를 비교해 보고 Operand 순서에 따른 성능의 차이에 대해 알아본다. 이때 Second operand 의 경우 lsl 명령어와 add 명령어로 구현하고, Multiplication operation은 mul 명령어를 사용하여 구현한다.

## 2. Design

### A. LSL 명령어

Bitstream을 왼쪽으로 SHIFT 하는 명령어로 한번 옮길 때 마다 2배씩 증가한다고 생각하면 된다. 만약 n번 옮긴다면 r0의 값은 $r0 * 2^n$이다. LSL 명령어는 단독으로 사용할 수도 있으며 기존 연산자와 묶어 사용할 수도 있다.

### B. Second operand

ADD 연산과 SHIFT 연산을 사용하여 곱연산을 구현하여 수행하는 것으로 이 때 SHIFT 연산의 경우 lsl 명령어를 사용하는데 이는 bit를 왼쪽 방향으로 shift 수행하는 것이다. 즉 값이 2배가 된다. 이를 add 명령어와 함께 사용하여 3을 곱연산 하려면 add r0, r0, r0, lsl #1 로 적을 수 있는데 이를 풀어 수학적으로 표현하면

$$r0 = r0 + r0 * 2 = r0 * (1 + 2)$$
$$= r0 * 3$$

임으로 r0에 3을 곱한 것과 같다. 이를 이용하여 3의 곱, 5의 곱, 등을 구현할 수 있으며 같은 원리로 뺄셈 연산인 rsb 명령어를 통해서 곱연산을 구현할 수 있다.

### C. MUL 명령어

곱연산을 실행하는 명령어로 예를 들어 MUL R1, R2, R3인 경우 R1 레지스터에 R2와 R3를 곱한 값을 저장한다. (R1= R2*R3)

### D. Multiplication operation

MUL 명령어를 사용하여 곱연산을 구현하여 수행하는 것이다. 말 그대로 곱셈만 사용한다.

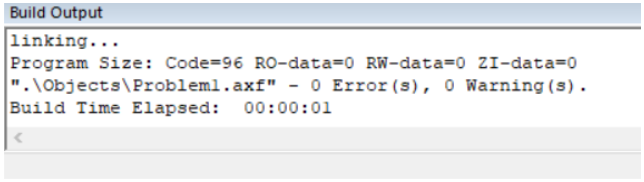
### E. Factorial

Factorial은 만약 1부터 n까지의 곱을 한다고 하면 n! 이라고 표현하며 예를 들어 1부터 10까지의 곱을 Factorial로 표현하면 10! 이다. 그리고 이를 연산하면 $1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 = 3628800$이다.

## 3. Conclusion

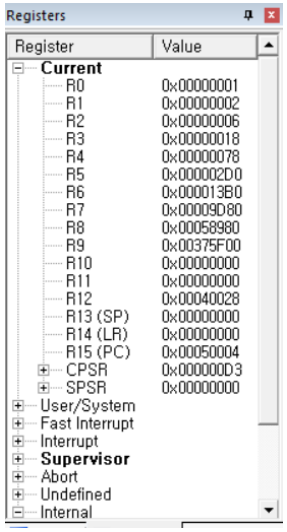
### A. Problem 1

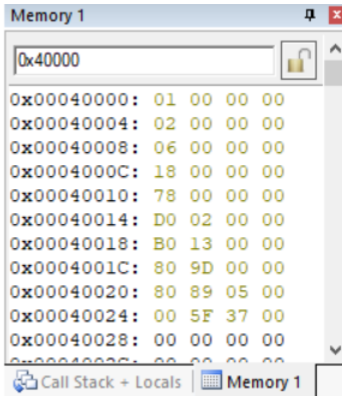
#### i. 코드 사이즈




코드 사이즈는 96이다.

#### ii. 레지스터 & 메모리

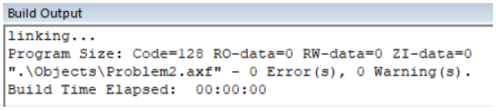



R0에는 1!=$1_{10}$=$1_{16}$, R1에는 2!=$2_{10}$ = $2_{16}$, R2에는 3!=$6_{10}$ = $6_{16}$ R3에는 4! = $24_{10}$ = $18_{16}$, R4에는 5! = $120_{10}$ = $78_{16}$, R5에는 6! = $720_{10}$ = $2D0_{16}$, R6에는 7! =$5040_{10}$ = $13B0_{16}$, R7에는 8! = $40320_{10}$ = $9D80_{16}$, R8에는 9! = $362880_{10}$ = $58950_{16}$, R9에는 10! = $3628800_{10}$ = $375f00_{16}$ 값이 저장되어 있으며, 후에 str 명령어를 통해 0x40000 메모리에 저장을 하였으며 little-endian 방식으로 저장되기 때문에 2개씩 역순으로 보이는 것을 위 사진을 통해 알 수 있다.

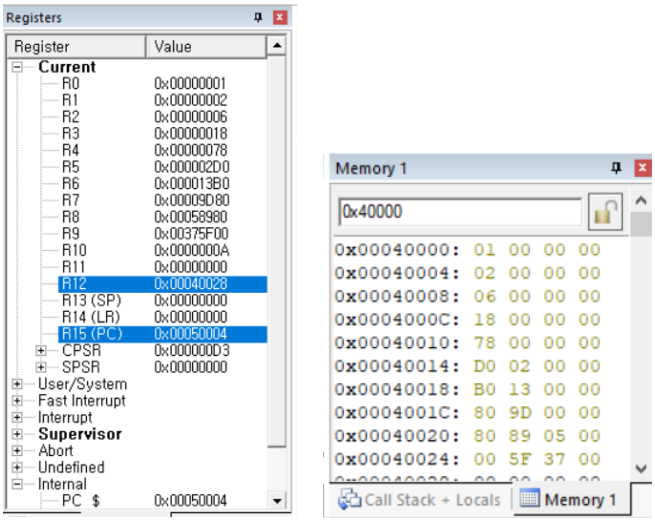
### B. Problem 2

#### i. 코드 사이즈




코드 사이즈 128이다.

#### ii. 레지스터 & 메모리




R10에는 MUL 명령어를 위해 1부터 10까지 1씩 늘려서 곱연산을 했으며 R0에는 1!=$1_{10}$=$1_{16}$, R1에는 2!=$2_{10}$ = $2_{16}$, R2에는 3!=$6_{10}$ = $6_{16}$ R3에는 4! = $24_{10}$ = $18_{16}$, R4에는 5! = $120_{10}$ = $78_{16}$, R5에는 6! = $720_{10}$ = $2D0_{16}$, R6에는 7! =$5040_{10}$ = $13B0_{16}$, R7에는 8! = $40320_{10}$ = $9D80_{16}$, R8에는 9! = $362880_{10}$ = $58950_{16}$, R9에는 10! = $3628800_{10}$ = $375f00_{16}$ 값이 저장되어 있으며, 후에 str 명령어를 통해 0x40000 메모리에 저장을 하였으며 little-endian 방식으로 저장되기 때문에 2개씩 역순으로 보이는 것을 위 사진을 통해 알 수 있다.

## 4. Consideration

이번 실습은 Second operand를 Add 연산과 LSL 연산을 이용하여 1부터 10까지의 곱인 Factorial을 구현하여 각 곱의 값을 저장하고 Factorial의 값을 저장했다. Multiplication operation을 MUL 연산을 통해 각 곱의 값을 저장하고, Factorial의 값을 저장했다.

Problem1과 Problem2는 결국 같은 Factorial을 구현하는 프로그램이지만 위의 결과처럼 코드 사이즈도 다르고 세부적인 속도도 다르다. LSL 명령어는 단순히 bit 단위를 왼쪽으로 옮기는 것이지만 MUL 명령어는 곱셈도 수행하고 저장해 주는 연산을 수행한다. 따라서 LSL 명령어를 사용하여 옮기는 shift 명령어가 더 빠를 것이라 생각된다.

대신 LSL 명령어는 만드는 사람이 각 곱을 구현하기 위해 어떤 방법으로 각 숫자를 표현할 것인지 생각해야 한다. 그러므로 Multiplication operation이 잘 짜여진 Second operand 보다 코드 사이즈가 크지만, 잘 생각하지 못하면 코드 사이즈가 더 커질 수 있을 것이라 생각한다.

## 5. Reference

1) 이형근 교수님/어셈블리프로그램설계및실습/광운대학교 컴퓨터정보공학부/2022