어셈블리 프로그래밍 설계 및 실습

실험제목: Second_Operand_&_Multiplication

실험일자: 2020 년 09 월 29 일 (화)

제출일자: 2020 년 10 월 05 일 (화)

학　　과: 컴퓨터공학과

담당교수: 이준환 교수님

학　　번: 2019202052

성　　명: 김 호 성

1. 제목및목적

A. 제목

Second_Operand_&_Multiplication

B. 목적

Second Operand 와 Multiplication 을 비교해 코드의 성능차이를 알아낸다.

2. 설계 (Design)

Problem1

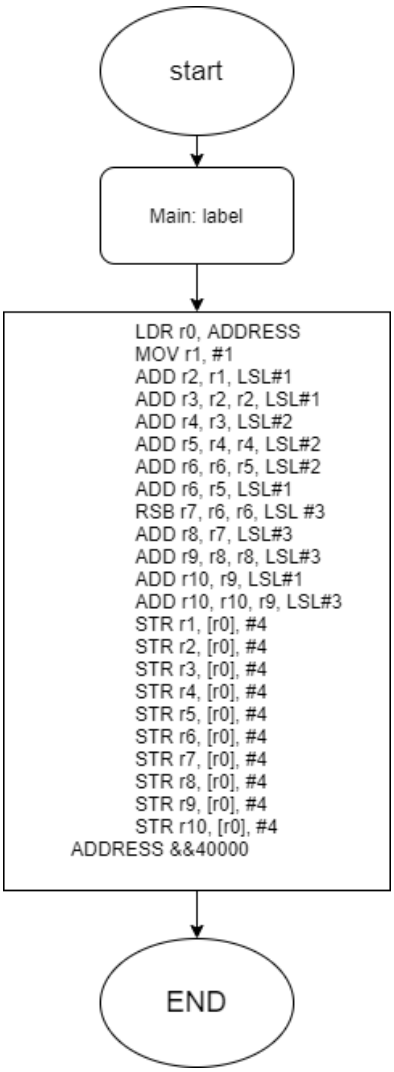
A. Pseudo code

Main

```
LDR R[0] ← ADDRESS; 0x00040000
MOV R[1] ← #1
ADD R[2] ← R[1](LSL#1) ; LSL #1 = *2
ADD R[3] ← R[2] + R[2](LSL#1)
ADD R[4] ← R[3](LSL #2); LSL #2 = *4
ADD R[5] ← R[4] + R[4](LSL#2)

ADD R[6] ← R[6] + R[5](LSL #2)
ADD R[6] ← R[5](LSL #1)

ADD R[7] ← R[6](LSL#3) – R[6] ;Revers Sub, LSL #3 = *8
ADD R[8] ← R[7](LSL #3)
ADD R[9] ← R[8] + R[8](LSL #3)

ADD R[10] ← R[9](LSL #1)
ADD R[10] ← R[10] + R[9](LSL #3)

STR R[1] → R[0], #4
STR R[2] → R[0], #4
STR R[3] → R[0], #4
```

```
        STR R[4] → R[0], #4
        STR R[5] → R[0], #4
        STR R[6] → R[0], #4
        STR R[7] → R[0], #4
        STR R[8] → R[0], #4
        STR R[9] → R[0], #4
        STR R[10] → R[0]

ADDRESS &&40000
        END
```

B. Flow chart 작성

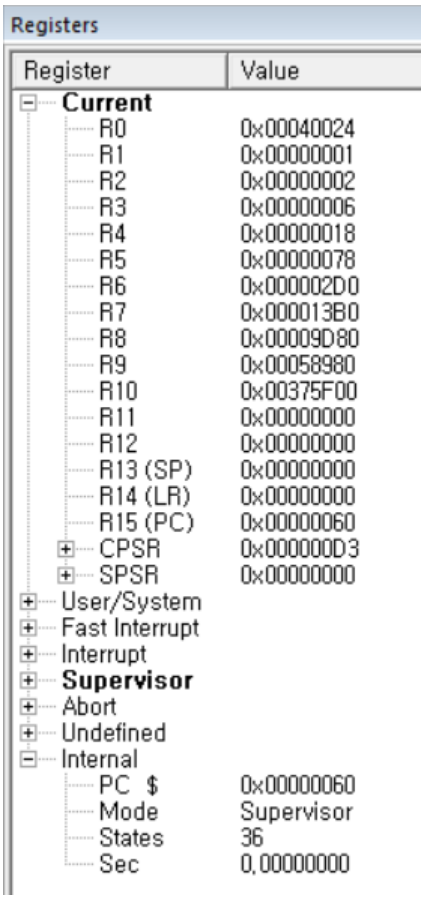


C. Result

결과를 보기 전 아래의 표를 참고하여 결과를 비교해보자.

10 진수	1	2	6	24	120	720	5040	40320	362880	3628800
16 진수	1	2	6	18	78	2D0	13B0	9D80	58980	375F00

Register 의 상태



메모리에 저장된 값들

```
0x00040000: 01 00 00 00 02 00 00 00 06 00 00 00 18 00 00 00 78 00 00 00 D0 02 00 00 B0 13 00
0x0004001B: 00 80 9D 00 00 80 89 05 00 00 5F 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

D. Performance

```
==========================================================

    Total RO  Size (Code + RO Data)                96 (   0.09kB)
    Total RW  Size (RW Data + ZI Data)              0 (   0.00kB)
    Total ROM Size (Code + RO Data + RW Data)       96 (   0.09kB)
```

Performance = Code Size * States =

위 사진에서 보이듯이 state =36, code = 96 으로,

결론: 3456

설계 (Design)

Problem2

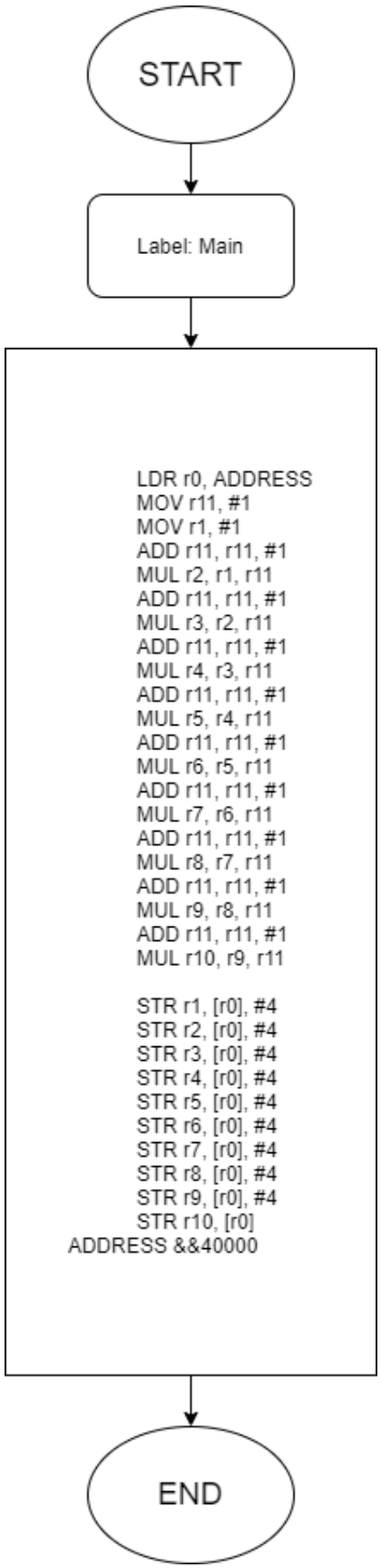
A. Pseudo code

Main

```
LDR R[0] ← ADDRESS ;0x40000
MOV R[11] ← #1
MOV R[1] ← #1

ADD R[11] ← R[11] + 1
MUL R[2] ← R[1] * R[11] ; 1*2

ADD R[11] ← R[11] + 1
MUL R[3] ← R[2] * R[11] ; 2*3

ADD R[11] ← R[11] + 1
MUL R[4] ← R[3] * R[11] ; 6*4

ADD R[11] ← R[11] + 1
MUL R[5] ← R[4] * R[11] ; 24*5

ADD R[11] ← R[11] + 1
MUL R[6] ← R[5] * R[11] ; 120*6

ADD R[11] ← R[11] + 1
MUL R[7] ← R[6] * R[11] ; 720*7

ADD R[11] ← R[11] + 1
MUL R[8] ← R[7] * R[11] ; 5,040*8

ADD R[11] ← R[11] + 1
MUL R[9] ← R[8] * R[11] ; 40,320*9
```

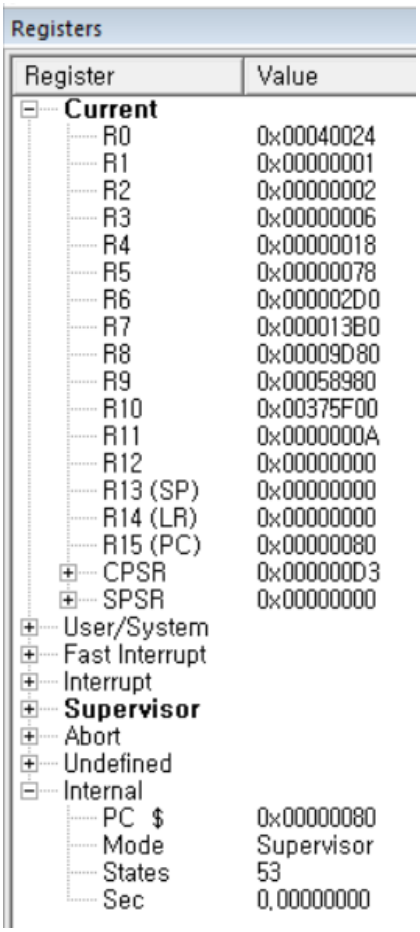
```
        ADD R[11] ← R[11] + 1
        MUL R[10] ← R[9] * R[11] ; 362,880*10

        STR R[1] → R[0], #4
        STR R[2] → R[0], #4
        STR R[3] → R[0], #4
        STR R[4] → R[0], #4
        STR R[5] → R[0], #4
        STR R[6] → R[0], #4
        STR R[7] → R[0], #4
        STR R[8] → R[0], #4
        STR R[9] → R[0], #4
        STR R[10] → R[0]
ADDRESS &&40000
        END
```

B. Flow chart 작성



C. Result

Register



Memory

```
0x00040000: 01 00 00 00 02 00 00 00 06 00 00 00 18 00 00 00 78 00 00 00 D0 02 00 00 B0 13 00
0x0004001B: 00 80 9D 00 00 80 89 05 00 00 5F 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

D. Performance

```
==========================================================

Total RO  Size (Code + RO Data)                 128 (   0.13kB)
Total RW  Size (RW Data + ZI Data)               0 (   0.00kB)
Total ROM Size (Code + RO Data + RW Data)       128 (   0.13kB)
```

Performance = Code Size * States =

128*53 = 6784

3. 고찰및결론

A. 고찰

Second Operand 에서 비트 연산을 위해 필요했던 식들

=> 비트 연산을 하기 위해 1, 2, 4, 8 의 배수들로 식을 정리했다. N!일 때 정리되는 값들이다.(A(1) = 1!, A(2) = 2!...)

10 진수	1	2	6	24	120	720	5040	40320	362880	3628800
16 진수	1	2	6	18	78	2D0	13B0	9D80	58980	375F00

A(1) = 1
A(2) = 2*A(1)
A(3) = A(2) + 2*A(2)
A(4) = 4*A(3)
A(5) = A(4) + 4*A(4)
A(6) = 4*A(5) + 2*A(5)
A(7) = 8*A(6) – A(6)
A(8) = 8*A(7)
A(9) = A(8) + 8*A(8)
A(10) = 2*A(9) + 8*A(9)

10 진수 기준으로 이 식에 맞게 비트연산을 해주면 된다. (비트연산이 2 번 들어가는 식의 경우 두번의 instruction 으로 나눠준다. [6! 과 10!])

B. 결론

Second operand 의 경우 비트연산을 필요로 하는 개념이기 때문에 곱셈을 쓰지 않기 위해선 2^n-1 (n = 1 부터 자연수)의 배수인 수들로 쪼개서 더했어야 했다. 그러다 보면 고찰의 A(6)이나 A(10)처럼 LSL 이 두 번 필요한 경우가 있는데, 이 경우에는 ADD 를 두 번으로 나누어서 계산해주면 된다.

메모리에 값이 저장될 경우 1byte(=8bit)단위로 저장되는데 이때 값이 이상하게 저장되는 것처럼 보일 수 있다. 그러나 현재의 경우 little endian 방식에서 코딩이 되어있고, 8bit 는 256 으로 16 진수로 변환 했을 때 0~256 까지 즉 0~FF 까지 저장할 수 있기에 정상적으로 저장된 것을 확인할 수 있다.

또한 Performance 를 비교해 보면, 3,456 vs 6,784 로 Second operand 방법이 더욱 성능이 좋다는 것을 확인할 수 있다.

4. 참고문헌

이준환교수님/디지털논리회로 2/광운대학교(컴퓨터정보공학부)/2020
공영호교수님/디지털논리회로 2/광운대학교(컴퓨터정보공학부)/2020