어셈블리프로그램 설계및실습 보고서

4주차 과제

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화요일 6, 7

학 번: 2019202050

성 명: 이강현

제 출 일: 2022.10.12(수)

1. Problem Statement

   Multiplication operation과 Second operand를 활용한 코드들의 성능을 비교해보고 Operand의 순서에 따른 성능의 차이에 대해 알아본다.

   Multiplication과 Second operand를 활용하여 코드를 구현해본다.

2. Design

   <Problem1>

   1. R0에 주소값 40000을 담는다.
   2. R1에 10을 넣는다. (10)
   3. R2에 R1과 R1을 왼쪽으로 3비트 이동하여 8배가 된 값을 더해서 넣는다.(10*9)
   4. R3에 R2를 왼쪽으로 3비트 이동하여 8배가 된 값을 넣는다.(10*9*8)
   5. R4에 R3를 왼쪽으로 3비트 이동하여 8배가 된 값에서 R3를 빼서 넣는다.(10*9*8*7)
   6. R4에 R4를 왼쪽으로 1비트 이동한 값을 넣고 그후 R5에 R4를 2비트 이동한 값을 넣는다. (10*9*8*7*6)
   7. R6에 R5와 R5를 왼쪽으로 2비트 이동한 값을 더해서 넣는다.(10*9*8*7*6*5)
   8. R7에 R6를 왼쪽으로 2비트 이동한 값을 넣는다.(10*9*8*7*6*5*4)
   9. R8에 R7과 R7를 왼쪽으로 1비트 이동한 값을 더해서 넣는다 (10*9*8*7*6*5*4*3)
   10. R9에 R8을 왼쪽으로 1비트 이동한 값을 넣는다.(10*9*8*7*6*5*4*3*2)
   11. R9값을 R0의 주소에 저장한다.

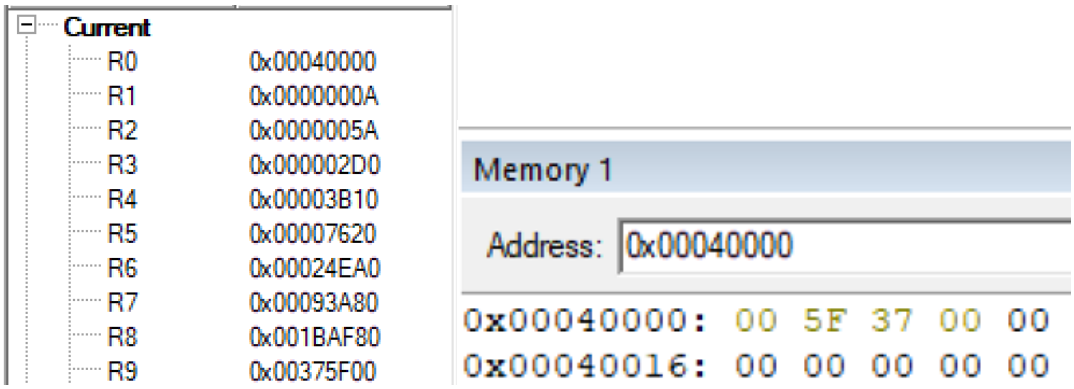
   <Problem2>

   1. R0에 주소값 40000을 담는다.
   2. R1에 값 10을 넣는다.(10)
   3. R2에 값 2를 넣는다.
   4. R3에 R1와 R2을 곱한 값을 넣는다.(10*2)
   5. R2를 1증가시킨다.
   6. R4에 R3와 R2를 곱한 값을 넣는다(10*3*2)
   7. R2를 1증가시킨다.

8. R5에 R4와 R2를 곱한 값을 넣는다.(10*4*3*2)
9. 이를 R2가 9가 될때까지 반복한다.
10. R10에 10!값이 저장된다. 이를 R0주소에 저장한다.(10*9*8*7*6*5*4*3*2)

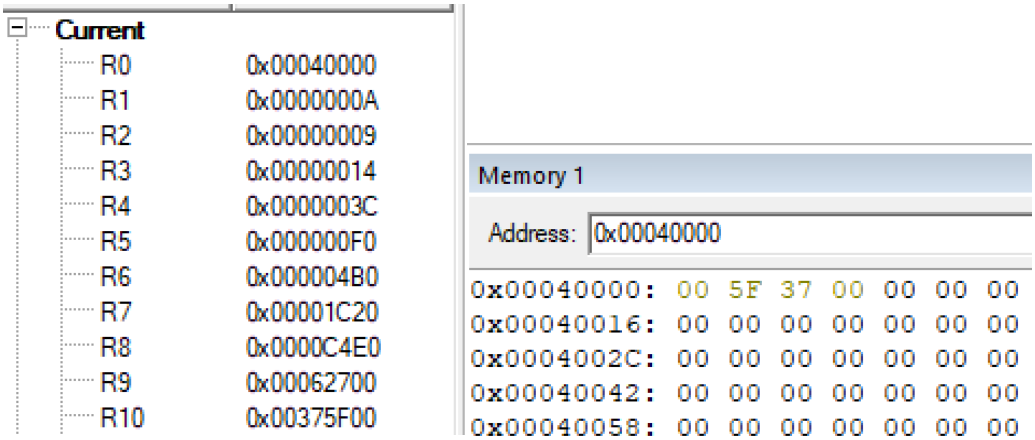
3. Conclusion

<Problem1>

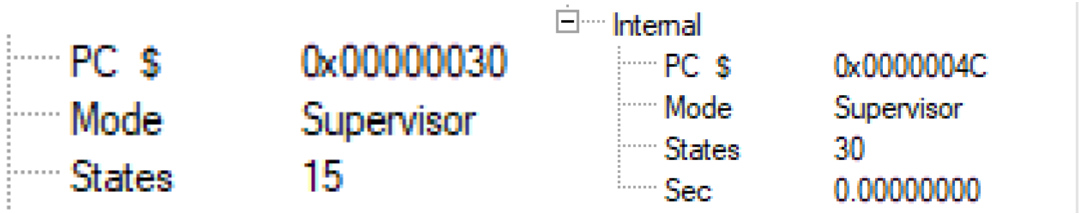


10부터 차근차근 9,8,7 순으로 곱해나가 최종적으로 r9에 10!의 값인 375f00(3628800)을 얻었다. 또한 40000의 주소값에 해당값이 저장되었음을 확인할 수 있다.

<Problem2>



문제 2번은 MUL instruction을 이용해서 레지스터에 결과값을 넣었다는 것을 알 수 있다. 2번은 R2의 값을 1씩 증감시켜 곱연산을 해주었기 때문에 끝자리부터 곱한 문제 1번과 결과는 다르지만 최종적으로 r10에 저장된 값은 375f00으로 1번의 값과 동일하다는 것을 알 수 있다.



둘의 성능을 비교한 결과 second operation을 한 연산이 state수를 반절로 줄인 것으로 보아 multiplication 연산보다 더욱 효율적인 연산이 가능하다는 것을 알 수 있었다.

4. Consideration

이번 실습은 어떻게 곱하기를 lsl과 add만으로 해낼 수 있을까에 대한 생각이 많이 필요했다. 예를 들면 10!을 연산하는 예제였기에 9를 곱하려면 8배한 것과 자기 자신을 더해줘야 한다는 것 그리고 8배는 왼쪽으로 3만큼 비트이동을 하면 얻을 수 있다는 것 이러한 생각을 해내는게 이번 과제의 핵심이었던 것 같다. Mul 명령어를 사용하는 것은 오히려 second oprand의 경우보다 생각해내는 것은 훨씬 간단했다. 단순히 하나씩 곱해가면 되었기 때문에 10부터 1까지 mul 명령어를 사용해주면 되었다. 하지만 state의 차이를 보았을 때 우리에겐 직관적으로 연산을 보여주어 간단하지만 프로그램의 측면에서는 더 무리가 되는 작업인 것 같다. 하지만 더욱 복잡한 연산처리를 해야할 때는 무리하게 비트이동을 이용하여 곱셈을 구현하려다 state수가 길어져 성능이 감소할 수도 있다고 생각된다. 따라서 쉬프트 연산을 통해 곱셈을 구현하는 것을 알고 상황에 맞게 코드를 구현한다면 효율적이고 어셈블리 언어를 더 잘 활용하는 것이라 생각했다.

5. Reference

이형근/어셈블리프로그램 설계 및 실습/광운대학교(컴퓨터정보공학부)/2022