어셈블리 프로그래밍 설계 및 실습

실험제목: Second Operand & Multiplication

실험일자: 2017년 09월 27일 (수)

제출일자: 2017년 10월 11일 (수)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습분반: 월 5, 수 6, 7

학 번: 2014722075

성 명: 이 동 준

1. 제목 및 목적 (3%)
   1. 제목

Second Operand & Multiplication

* 1. 목적

어셈블리를 통해 Factorial을 설계하는데 second operand와 multiplication operation을 활용하여 설계를 해보고 두 개의 성능차이에 대해 비교해본다.

1. 설계 (Design) (50%)
   1. Pseudo code

\* problem1

r11 = address (0x40000)

r1<-1 ; 1!

store in r11

r2<-r1, LSL #1 ; 2!

store in r11

r3<-r2, LSL #1

r3 = r3 + r2 ; 3!

store in r11

r4<-r3, LSL #2 ; 4!

store in r11

r5<-r4, LSL #2

r5 = r5 + r4 ; 5!

store in r11

r6<-r5, LSL #2

r6 = r6 + r5, LSL #1 ; 6!

store in r11

r7<-r6, LSL #3

r7 = r7 – r6 ; 7!

store in r11

r8<-r7, LSL #3 ; 8!

store in r11

r9<-r8, LSL #3

r9 = r9 – r8 ; 9!

store in r11

r10<-r9, LSL #3

r10 = r10 + r9 ; 10!

store in r11

\* problem2

r11 = address (0x40000)

r0 = value (1,2,3,4,5,6,7,8,9,10)

r12<-r0(1) ; 1!

store in r11

r12<-r0(2)

r2 = r1 x r12 ; 2!

store in r11

r12<-r0(3)

r3 = r2 x r12 ; 3!

store in r11

r12<-r0(4)

r4 = r3 x r12 ; 4!

store in r11

r12<-r0(5)

r5 = r4 x r12 ; 5!

store in r11

r12<-r0(6)

r6 = r5 x r12 ; 6!

store in r11

r12<-r0(7)

r7 = r6 x r12 ; 7!

store in r11

r12<-r0(8)

r8 = r7 x r12 ; 8!

store in r11

r12<-r0(9)

r9 = r8 x r12 ; 9!

store in r11

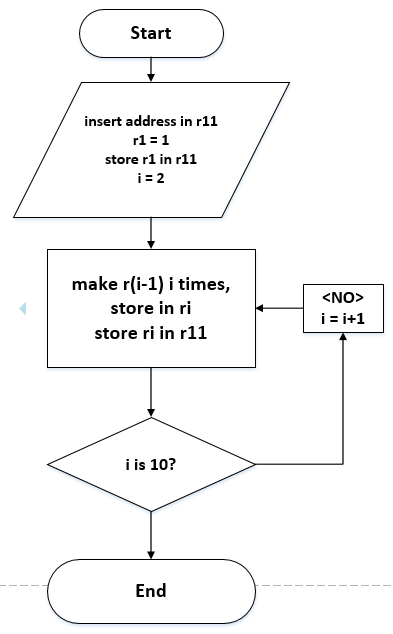
r12<-r0(10)

r10 = r9 x r12 ; 10!

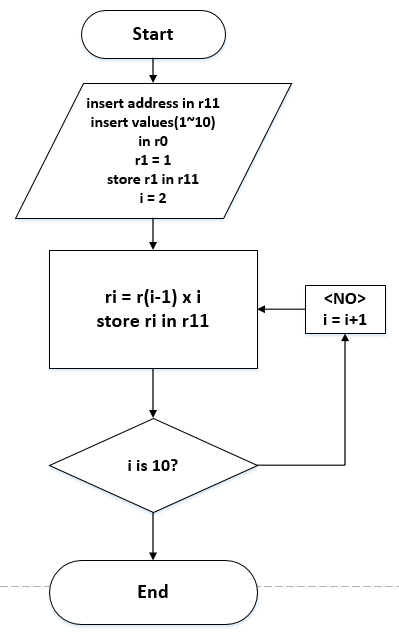
store in r11

* 1. Flow chart 작성

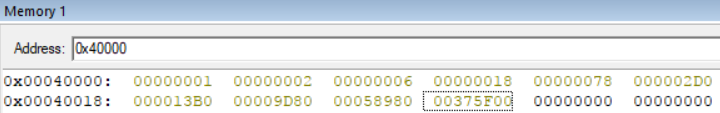
\* problem1



\* problem2



* 1. Result



* 1. Performance

\* problem1

state = 40, codesize = 120bytes

\* problem2

state = 87, codesize = 176bytes

->확실히 second operand를 이용한 것이 성능에서 앞서는 걸 볼 수 있다.

1. 고찰 및 결론
   1. 고찰 (35%)

이번 실습에서는 Factorial을 second operand와 multiplication operation을 통해서 설계를 해보았다. second operand에서는 단순히 각 값들은 곱해주는 것이 아닌 logical shift 을 이용하였는데 처음에는 어떻게 해야 좋은 프로그램을 만들 수 있을까 고민을 하였는데 강의자료를 참고하면서 완성하였다. state와 codesize를 보더라도 state에서는 두배 이상의 차이가 났고, score를 계산하여 보면 상당한 차이가 나는 걸 눈으로 확인할 수 있었다. 어떠한 명령어를 사용하고 어떠한 알고리즘으로 코드를 완성하냐 따라 프로그램의 성능이 달라지는 것을 다시 한번 확인할 수 있었다. 어셈블리 코드를 활용하는 면에서 어느정도 숙달이 되고 있지만, 개념적으로는 부족하다고 생각하여 개념에 있어서 복습이 반드시 필요하다.

* 1. 결론 (10%)

logical shift를 이용하면 어떠한 수를 곱하는데 있어서 좀 더 유용하고 좋은 성능으로 가능하다. 특히 명령어 중 MUL은 다른 명령어에 비해 많은 state를 사용하는데, 다른 방법으로 multiplication을 수행 할 수 있어서 더 나은 방법을 선택하는 것이 바람직하다.

1. 참고문헌 (2%)

이형근교수님 – 어셈블리프로그램 설계 및 실습 – 강의자료