어셈블리프로그래밍설계및실습 보고서

실험제목: Data transfer to or from Mem

실험일자: 2016년 09월 22일 (목)

제출일자: 2016년 09월 28일 (수)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습분반: 화5, 목6,7

학 번: 2013722095

성 명: 최 재 은

1. 제목 및 목적
   1. 제목

Data transfer to or from Mem

* 1. 목적

어셈블리의 기본 명령어 및 조건부 실행을 사용함으로써 숙지를 한다.

메모리를 저장하고 읽어 들이는 방법을 연습하고 익힘으로써 프로그램

구현 시 메모리가 어떻게 저장되는지 확인한다.

1. 설계 (Design)
   1. Pseudo code

**1) exercise1**

R0에 10을 저장

R1에 10을 저장

두 레지스터를 비교

R0이 더 작으면 r0를 r2에 저장

더 크면 r1을 r2에 저장

서로 같으면 두 레지스터의 합을 r0에 저장

**2) exercise2**

r0에 1을 저장

r1에-1을 저장

r2에 8을 저장

r0 와 0을 비교

r0>0이면 r3에 r0 저장

r1 와 0을 비교

r1>0 이면 r1+r3을 r3에 저장

r2와 7을 비교

r2<=7이면 r2+r3를 r3에 저장

r3와 0을 비교

서로 같으면 r0를 r3에 저장

**3) exercise3**

R0에 16, r1에 10, r2에 08, r3에 34 저장

r4에 TEMPADDER의 주소값 저장

r0의 값을 r4의 주소값 1000번지에 저장, r4를 2증가

r1의 값을 r4의 주소값 1002번지에 저장, r4를 2증가

r2의 값을 r4의 주소값 1004번지에 저장, r4를 2증가

r3의 값을 r4의 주소값 1006번지에 저장

**4) Problem1**

R0에 5, r1에 10, r2에 15, r3에 10(0A)저장

R4에 TEMP 주소값 저장

R9에 r4의 값 저장(백업)

R4주소값을 통해 찾아간 주소값에 r0,r1,r2,r3의 값 저장

R5,r6,r7에 메모리에 있는 값을 읽어들임.

R3,r2 비교

R2가 r3보다 크면 r5에 1을 넣고 1004번 메모리에 저장

R3,r0비교

R3가 r0보다 크면 r5에 2를 넣고 1005번 메모리에 저장

R3,r1 비교

R3=r1이면 r5에 3을 넣고 1006번 메모리에 저장

**5) Problem2**

R0에 1, r1에 2, r2에 3, r3에 4를 저장

R4에 temp의 주소값 저장

R7에 r4값 저장(백업)

R0,r1,r2,r3를 순차적으로 메모리에 저장

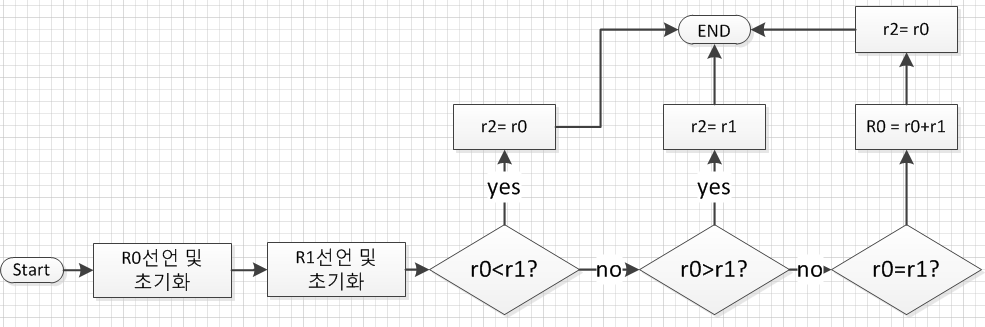
R5에 r7주소값의 memory들을 읽어들임(little-endian)

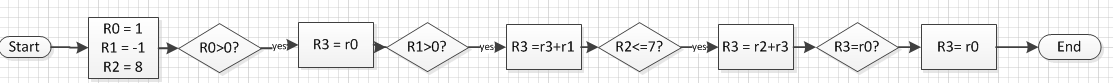
R4에 r7값 저장

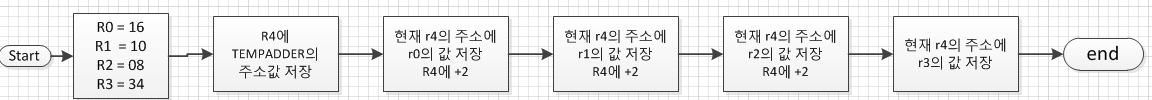
R3,r2,r1,r0를 순차적으로 메모리에 저장

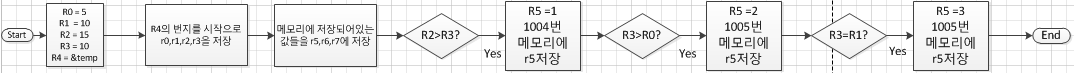
R6에 r7주소값의 memory들을 읽어들임(big-endian)

* 1. Flow chart 작성

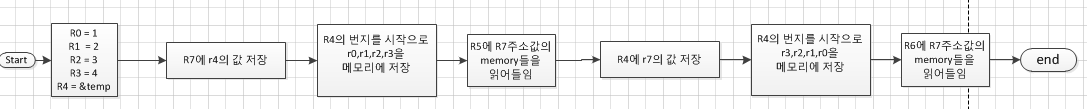
1) exercise.1

 2)exercise.2

 3)exercise.3

 4)Problem1

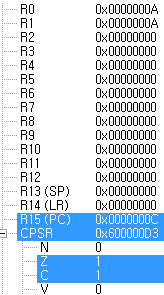
5)Problem2

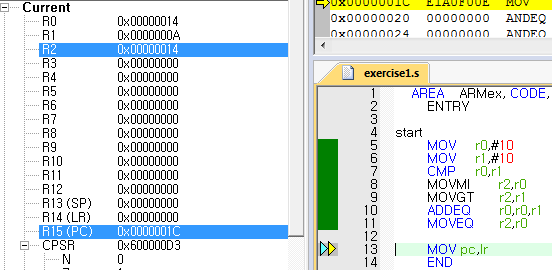


* 1. Result

1)exercise1

- r0, r1에 값이 저장된 것을 확인할 수 있습니다.



- 10+10해서 20의 값(H:14)이 r2에 저장된 것을 확인할 수 있습니다.

2)exercise2

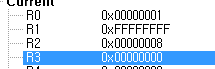
-레지스터에 초기값들이 저장된 상태입니다.



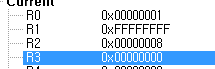
- r0와 0을 CMP해서 r0값이 r3에 저장된 상태입니다.



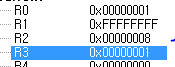
- r1과 0을 CMP해서 r1+r3가 r3에 저장된 상태입니다.



- r2와 7을 CMP한후 r2가 7보다 크기 때문에 r3값이 변하지 않았습니다.



- r3가 0과 같으면 r0를 r3에 넣어주는 문장의 결과입니다.

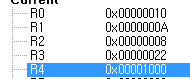


3)exercise3

- 메모리 접근 권한을 위한 memory.ini 참조



- 각 register에 초기값들과 접근할 메모리의 주소 값이 저장되었습니다.



- 0x00001000을 시작으로 한 칸씩 건너뛰어 값들이 메모리에 저장되었습니다.

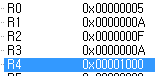


4) problem1

- 메모리 접근 권한 부여를 위한 memory.ini파일 참조



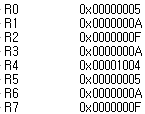
- register에 각 초기값들과 접근할 메모리의 주소값들이 저장되었습니다.



- 0x00001000을 시작으로 레지스터의 값들이 memory에 저장되었습니다.



- 메모리에 있던 값들을 r5~r7에 읽어들였습니다.

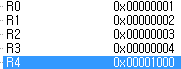


- 조건문을 통해 저장되었던 r5의 값들을 memory(1004~1006)에 저장시켜보았습니다.



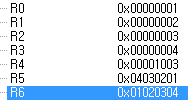
5) Problem2

- register에 각 초기값들과 접근할 메모리의 주소값들이 저장되었습니다.



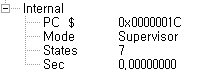
- r5에 little-endian, r6에 big-endian으로 memory값이 저장되었음을 확인할

수 있습니다.



* 1. Performance

1) exercise1

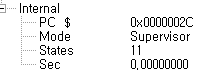


- code size는 32이고 state는 7개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 1568이 나오게 됩니다.

2) exercise2





- code size는 48이고 state는 11개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 5808입니다.

3) exercise3





- code size는 44이고 state는 15개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 9900입니다.

4) Problem1



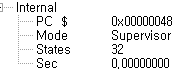


- code size는 100이고 state는 40개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 160000입니다.

5) Problem2





- code size는 76이고 state는 32개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 77824입니다.

1. 고찰 및 결론
   1. 고찰

디버그를 수행하면서 자꾸만 error 65가 뜨면서 무한 루프를 돌듯이 커서가 넘어가지않았는데 이를 mov pc,lr을 추가함으로써 해결하였습니다. 조건부 실행 명령어의 의미를 제대로 이해하지 못해서 혼동이 많았습니다. 조건부 명령어들의 의미를 이해하지 못한 상태여서 그런지 가져다 쓰는 레지스터들도 이상한 경우도 있었습니다. 이를 해결하는데 주위의 도움을 받았습니다.

* 1. 결론

조건부 명령어를 통해서 프로그램의 flow를 수행할 수 있을 줄 알았는데 실상 해보고 나니 Branch를 배워야 c,c ++에서처럼 프로그램을 구현할 수 있지 않을까 싶습니다.

C, c++에서의 if문 처럼 조건부 실행을 할 수 있게 된다면 좀더 register와 메모리를 효율적으로 관리 및 사용할 수 있을 것 같습니다.

1. 참고문헌

- 디버그가 끝나지 않는 현상을 해결하는데 13 김현지가 도움을 주었습니다.

- little/big endian의 개념을 이해하는데 13 김현지, 13 김동민이 도움을 주었습니다.