어셈블리프로그래밍설계및실습 보고서

실험제목: Control flow & Data processing

실험일자: 2016년 09월 29일 (목)

제출일자: 2016년 10월 05일 (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

Control flow & Data Processing

* 1. 목적

Flow를 control할수 있게 하는 Opcode를 배우고 이에 대한 사용법을 익히고 연습해봄으로써 프로그램의 state를 효율적으로 통제할 수 있는 능력을 키운다. Branch의 개념을 이해하고 이를 사용하는 방법을 익혀 전체 코드를 효율적으로 동작하게 한다.

1. 설계 (Design)
   1. Pseudo code

**1) problem1**

R0~r3까지 임의의 값을 저장

R0에 r0-r1의 값 저장

R1에 r3^2의 값 저장

R2에 r2+r1의 값 저장

R0에 r0+ r2의 값 저장

R1에 temp(40000000)의 주소값 저장

R1의 주소번지에 r0 저장

**2) problem 2**

R0에 value1의 메모리값(Helloword) 저장, r4에 temp메모리 번지 저장

loop선언

R3에 r0주소가 가리키는 번지에 저장된 값을 1byte 읽어들이고 r0의 주소 1증가

R3와 0을 비교

R3값이 0이 아니면 r2(count)를 1 증가시키고 loop로 점프

R3가 0이면 r2를 r4 메모리 번지에 저장

**3) problem3\_1**

R0에 1 저장, r2에 temp 메모리 번지 저장

loop선언

R0가 11인지 비교

아니라면 r1에 r0를 더하고 r0에 1을 더한 뒤 loop로 점프

맞으면 r1을 r2메모리 번지에 저장

**4)problem3\_2**

R0에 1 저장, r5에 temp메모리 번지 저장

loop선언

R1에 r0+1 저장

R0가 10인가?

아니라면 r0에 1을 더하고 loop로 점프

R2에 r0\*r1값을 저장

R2를 왼쪽으로 한 비트 이동시킴( /2 연산)

R5메모리번지에 r2 저장

**5) problem3\_3**

R0 에 1저장, r2에 temp메모리 번지 저장

R1에 r0값을 더함 }

---> 9번 반복

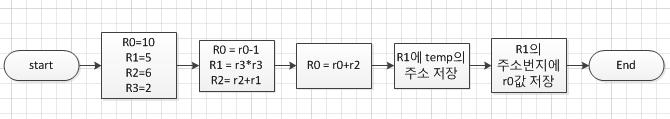
R0에 1 더함 }

R1에 r0값을 더함

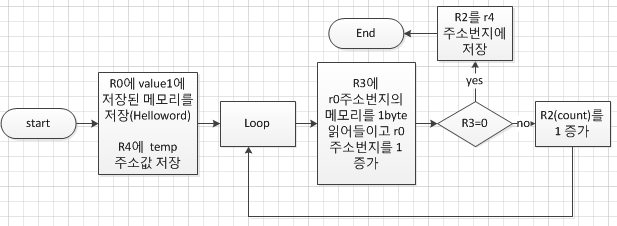
R1을 r2메모리 번지에 저장

* 1. Flow chart 작성

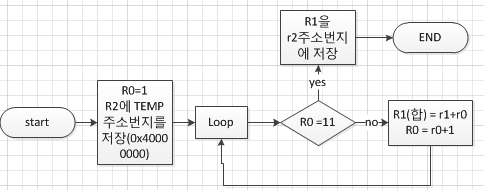
**1) problem1**



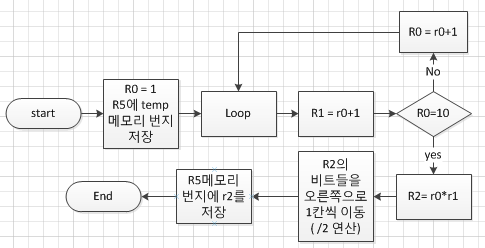
**2) problem2**



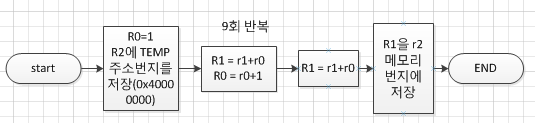
**3) problem3\_1**



**4) problem3\_2**



**5) problem3\_3**



* 1. Result

**1) problem1**

- 레지스터에 임의의 입력값을 저장함.



- 연산결과를 r0에 저장함



- r0를 메모리에 저장

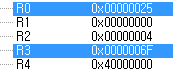


**2) problem2**

- value1에 들어간 helloword가 0x20번지에 저장되었습니다.



- 0x20번지 메모리부터 시간된 순회에서 count값(r2)가 증가하고 있습니다.



- 문장의 길이는 다음과 같이 계산되어 저장되었습니다.



**3) problem3\_1**

- 연산 결과화면



- 메모리에 잘 저장되었습니다.



**4) problem3\_2**

- 연산 결과화면



- 메모리에 저장



5) problem 3\_3

- 연산 결과화면



- 메모리에 저장된 사진



* 1. Performance

1) problem1





- code size는 48이고 state는 15개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 10800이 나오게 됩니다.

2) problem2





- code size는 52이고 state는 86개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 384592이 나오게 됩니다.

3) problem3\_1





- code size는 36이고 state는 70개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 176400이 나오게 됩니다.

4) problem3\_2





- code size는 44이고 state는 67개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 197516이 나오게 됩니다.

5) problem3\_3





- code size는 96이고 state는 25개입니다. Score는 code size \* state^2이므로

이 프로그램의 스코어는 60000이 나오게 됩니다.

1. 고찰 및 결론
   1. 고찰

나눗셈 연산을 하기 위해서 opcode를 찾아보는 바보 같은 짓을 하다가 bit shifting을

하면 곱셈 / 나눗셈을 할 수 있다는 것을 상기하고 LSR을 사용하여 나눗셈을 해주었

습니다.(problem3\_2)

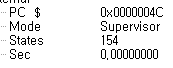
-EXERCISE : BRANCH





- EXERCISE : conditional execution





- 위의 두 경우를 비교해 보면 조건명령어를 사용했을 경우의 state가 더 크다는 것을

확인할 수 있습니다. Branch를 사용하게 되면 조건에 해당하지 않는 경우는 jump를

통해 생략이 되지만 조건명령어를 사용하게 되면 CMP를 통해 확인된 조건에의 여하를

모두 확인하기 때문에 생략이 없다고 볼 수 있습니다. 이 때문에 score가 각각

B:2387392 / C:2656192로 큰 차이를 보이며 BRANCH를 사용하는 것이 더 효율적이라

는 것을 볼 수 있습니다.

* 1. 결론

Problem3을 3개의 방식으로 진행해보았습니다. 3번째 방법의 경우는 모든 경우의 수를

직접 입력하는 것으로 flow를 control하는 듯한 느낌은 아니었습니다. 가장 긴 code size를 갖는 반면에 가장 적은 state를 갖습니다. 하지만 이 방법이 복잡하고 경우의 수가 많은 case에서는 사용될 수 없겠다고 생각했습니다. 3번째 방법을 loop로 짧게 만든 것이

1번째 방법인 것 같다고 생각합니다. 코드의 길이가 현저하게 줄지만 Branch를 통해 점프하게 되어 비교연산을 하기 때문에 state는 많이 차이가 납니다. 2번째 방법은 공식을 사용하여 합을 계산하는 방식인데 1번째 방법과의 비교를 위해서 1~10까지의 경우를 모두 확인하였습니다. 이때 state의 수는 서로 비슷하지만 code size가 늘어나게 됩니다.

4. 참고문헌