어셈블리프로그램 설계및실습 보고서

과제 주차: 3주차

학 과: 컴퓨터정보공학부

담당교수: 이형근 교수님

실습분반: 화요일 6, 7

학 번: 2019202050

성 명: 이강현

제 출 일: 2022.09.30(수)

1. Problem Statement

Memory로부터 값을 받아 이를 이용해서 원하는 연산을 수행한다. Assembly code를 작성할 때 performance를 고려하며 작성한다. Branch 명령어와 conditional execution의 차이를 알아본다. Loop를 사용하여 조건문의 분기를 확인해본다. Memory를 할당하여 문자열과 숫자배열을 저장해본다.

2. Design

<Problem1>

문자열 비교 후 값을 넣어줄 메모리 0x00004000을 r0에 넣어준다.

DCB명령어를 이용하여 arr1,arr2에 각각 문자열을 넣어준 후 Register r5,r6에 각각 문자열들의 주소값을 넣는다.

문자열 비교 후 넣어줄 상수 10과 11을 각각 r1,r2에 넣어준다.

루프문을 통해 메모리에 저장된 문자열들을 각각 1바이트씩 r3,r4에 저장해 비교연산을 실행한다. 같으면 equal로 다르면 notequal로 이동하게 끔 분기를 설정한다.

equal이 실행되면 한번더 비교하여 숫자가 0이 아닌지 체크한 후 문제에서 설정한 값 10을 r0에 저장되어있는 주소값으로 저장해주고 notequal이 실행되면 11을 저장한다.

<Problem2>

문자열 비교 후 값을 넣어줄 메모리 0x00004000을 r0에 넣어준 후 DCD명령어를 통해 10부터 1까지의 값들을 순차적으로 array변수에 넣어준다.

array의 주소값은 r1에 저장해두고 r2에는 r1을 뒤로 9칸 이동한 곳의 숫자를 넣어준다.

루프문을 활용하여 r1을 앞으로 1칸씩 이동해가며 r0에 저장된 메모리 주소로 저장한다.

<Problem3-1>

r0에 메모리 0x00004000을 load한다.

r1에 1을 저장한 뒤, LSL과 ADD를 통해 11을 만든 후 r2에 저장한다.

r4에 11부터 29까지의 값을 더해간다.

루프문을 활용해 r1에 2씩 더해가고 r1과 r2를 r3에 저장한다.
r4에 r3의 값을 더해주고 r3가 29가 되면 루프문을 탈출한다.

<Problem3-2>

r0에 메모리 0x00004000을 load한다.
r1에 1의 값을 저장해 LSL과 ADD연산을 통해 10을 만든다. 이후 r2에 저장한다.
n(n+10)에서 n이 10이기 때문에 r3에 10+10을 연산한 값을 저장해 준다.
MUL을 통해 10*20을 계산한 후, r4에 저장해준다.
r4에 저장된 결과 값을 메모리에 저장해준다.

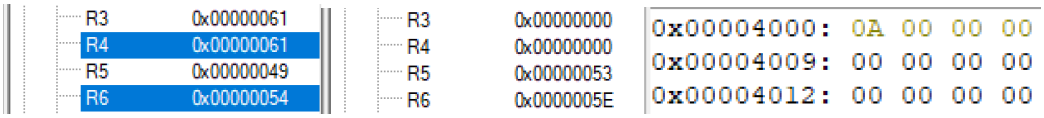
<Problem3-3>

r0에 메모리 0x00004000을 load한다.
11,13~27,29까지의 배열을 array에 저장한후 주소값을 r1에 저장한다.
루프를 사용하지 않고 배열에 4바이트 단위로 접근하여 값을 r3에 저장한다.
메모리에 r3에 저장된 값을 저장한다.

3. Conclusion

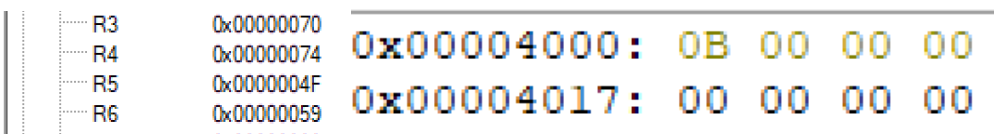
<Problem1>

apple_tree와 apple_tree비교



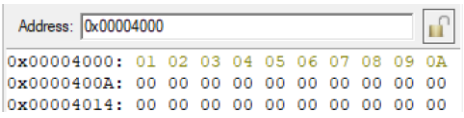
R3와 R4값이 바이트단위로 계속 같기때문에 0A가 저장되었고 이어서 0이라는 문자를 만나게 되었을 때 반복문을 탈출하고 0A가 저장된 채로 코드가 마무리되는 것을 확인할 수 있다.

apple_pie와 apple_tree비교



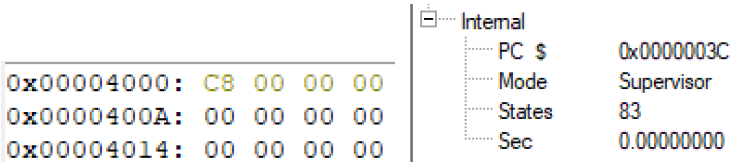
pie와 tree 부분이 다른 것이 R3,R4값이 서로 다름을 통해 보여지고 그로 인해 0B가 저장된 후 다른 문자열을 만났으므로 함수를 탈출한다.

<Problem2>



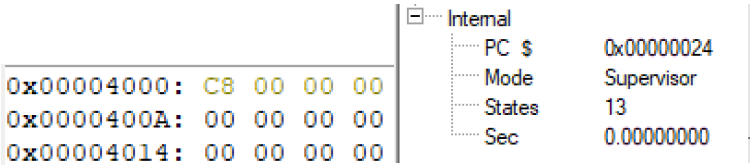
01,02~09,0A로 순서가 올바르게 저장된 것을 확인할 수 있다.

<Problem3-1>



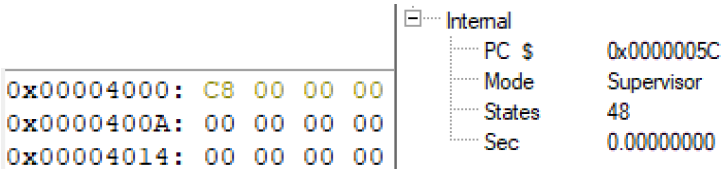
C8(200)이라는 수를 정확하게 얻었다.

<Problem3-2>



C8(200)이라는 수를 정확하게 얻었다.

<Problem3-3>



C8(200)이라는 수를 정확하게 얻었다.

Problem의 1은 루프를 활용한 방법, 2는 일반화된 식을 활용한 방법, 3은 unrolling을 활용한 방법이다. 실행된 횟수가 전부 다르며 일반화된 식을 사용한 2번 케이스는 최적화된 식을 계산만 하면 되므로 states수가 현저히 적다. 그 다음으로 unrolling을 사용한 것은 루프문을 돌지 않게 순차적으로 제작하였으며 코드의 길이가 많아지고 중복도가 높아졌지만 1번 케이스에 비해 states수는 감소하였다.

1번케이스는 가장 states 수가 높게 생겼는데 이는 branch를 활용하여 코드를 여러번 반복하기 때문이라고 생각된다.

4. Consideration

이번 예제를 통해 메모리를 원하는 바이트단위로 나누어 할당하는 방법을 배웠다. 이는 문자열과 숫자배열을 메모리에 쉽게 저장하는 법을 학습할 수 있는 좋은 기회였다. 또한 루프문을 사용하여 반복되는 계산을 루프를 활용하여 코드의 중복성을 줄이고 간단하게 만들 수 있다는 것을 배웠다. 하지만 단순히 루프만을 사용하여 코드를 구성하는 것은 무조건적으로 장점만 존재하는 것이 아님을 알 수 있었다. Branch와 unrolling을 최대한으로 사용해 루프를 사용하지 않은 각각 3-1과 3-3을 비교하여 보았을 때 3-3은 코드는 늘었으나 states수가 3-1에 비해 반절 가까이 감소한 것으로 보아 연산이 완료되는데 까지의 시간을 줄일 수 있었다. 단순한 계산이었기에 conditional execution방법이 branch방법보다 우수하다고 판단할 수는 없다. 따라서 코드의 중복도를 루프로 적당히 줄이지만 unrolling방법을 적당히 고려하여 코드를 구성하면 더더욱 성능을 높일 수 있을 것이라는 결론을 얻었다.

5. Reference

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