어셈블리 프로그래밍 설계 및 실습

실험제목: Block Data Transfer & Stack

실험일자: 2017년 10월 12일 (목)

제출일자: 2017년 10월 25일 (수)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 화 5, 목 6, 7

학 번: 2012722028

성 명: 장 한 별

1. 제목 및 목적 (3%)
   1. 제목

Block Data Transfer & Stack

* 1. 목적

Block data를 transfer 하는 즉, Register 와 Memory 사이 Block 단위의 Data를 저장하고 불러오는 기능을 배우고 이해하고 그 사용법을 익히도록 한다. 또한 Stack의 기본적인 구조를 이해하고 stack을 이용하여 data를 저장하고 불러오는 방법에 대해서도 이해하고 그 사용법을 익히도록 하는 데에 목적이 있다.

1. 설계 (Design) (50%)

<Problem.1>

* 1. Pseudo code

|  |
| --- |
| {  R0 = 0  R1 = 1  R2 = 2  R3 = 3  R4 = 4  R5 = 5  R6 = 6  R7 = 7  Sp = &40000  Store r2 into Sp, Sp=&40004  Store r0, r3 into Sp, Sp=&4000C  Store r5, r6, r7 into Sp, Sp=&40014  Store r1, r4 into Sp, Sp=&4001C  Load sp to r0-r7  } |

* 1. Flow chart 작성

|  |
| --- |
|  |

위 그림은 Problem 1 의 Flow chart 이다.

* 1. Result

|  |  |
| --- | --- |
|  | C:\Users\장한별 (Stanley)\Desktop\1.메모리.JPG |

위 그림 중 왼쪽 그림은 Stack에 저장되어 있는 값들을 Full Ascending 방법으로 r0~r7 에 저장한 뒤의 Register 화면 이다. Problem.1. 에 주어진 순서대로 정확하게 load 되어있음을 확인할 수 있다.

오른쪽 그림을 보면 Memory 에 Register 의 값들이 올바르게 Store 된 것을 확인할 수 있다.

* 1. Performance

|  |  |
| --- | --- |
| C:\Users\장한별 (Stanley)\Desktop\1.co.JPG | C:\Users\장한별 (Stanley)\Desktop\1.인.JPG |

위 그림을 보면 Code size 는 64 bytes, State 는 33인 것을 확인할 수 있다. 따라서 Score = 64 \* 332 = 69,696 이다.

<Problem.2>.

1. Pseudo code

|  |
| --- |
| {  R0 = 10  R1 = 11  R2 = 12  R3 = 13  R4 = 14  R5 = 15  R6 = 16  R7 = 17  R9 = 160  R12 = &40000  Store from r0 to r7 into sp, sp!  doRegister()  {  R0 = r0 + 0  R1 = r1 + 1  R2 = r2 + 2  R3 = r3 + 3  R4 = r4 + 4  R5 = r5 + 5  R6 = r6 + 6  R7 = r7 + 7  R8 = r0 + r1  R8 = r8 + r2  R8 = r8 + r3  R8 = r8 + r4  R8 = r8 + r5  R8 = r8 + r6  R8 = r8 + r7  }  doGCD()  {  While(1)  {  If (r8 > r9)  R8 = r8 + r9  else if (r8 < r9)  r9 = r9 + r8  if (r8 == r9)  break  }  }  R7 = r7 + 24  R6 = r6 + 22  R5 = r5+ 20  R4 = r4 + 18  R3 = r3 + 16  R2 = r2 + 14  R1 = r1 + 12  R0 = r0 + 10  Store r9 into r 12  Store from r0 to r7 into r 12  } |

1. Flow chart 작성

|  |
| --- |
| C:\Users\장한별 (Stanley)\Desktop\2.flow.JPG |

위 그림은 Problem.2.의 Flow chart 이다.

1. Result

|  |  |
| --- | --- |
| C:\Users\장한별 (Stanley)\Desktop\2. 격ㄹ과.JPG | C:\Users\장한별 (Stanley)\Desktop\2.메.JPG |

위 그림 중 왼쪽 그림을 보면 함수 doRegister() 를 호출하기 전의 Register 값들과 호출한 후의 Register 의 값들의 합이 정확하게 구현되었음을 확인할 수 있다.

오른쪽 그림을 보면 함수 doRegister() 로 구현한 r0 부터 r7의 합, 즉 136과 160의 최대 공약수인 8이 Memory에 Store 된 것을 확인할 수 있다. 그 후 함수 doRegister() 를 호출하기 전의 Register 값들과 호출한 후의 Register 의 값들의 합이 정확하게 Store 된 것을 확인할 수 있다. 이때 GCD 를 이용해 얻은 최대공약수와 그 후 Register 들의 합들을 구별하기 위해 주소값에 4byte 를 더해주어 한 눈에 구별할 수 있도록 해주었다.

1. Performance

|  |  |
| --- | --- |
| C:\Users\장한별 (Stanley)\Desktop\2.사이즈.JPG | C:\Users\장한별 (Stanley)\Desktop\2. st.JPG |

위 그림을 보면 Code size 는 212 Bytes, States 는 140 인 것을 확인할 수 있다.

따라서 Score = 212 \* 1402 = 4,155,200 이다.

1. 고찰 및 결론
   1. 고찰 (35%)

이번 실습시간에는 예제를 통해 Block data를 transfer 하는 즉, Register 와 Memory Block 단위의 Data를 저장하고 불러오는 기능을 배우고 이해하고 그 사용법을 익히는 데에 목적이 있었다. 또한 Stack의 기본적인 구조를 이해하고 Stack을 이용하여 Data를 저장하고 불러오는 방법에 대해서도 이해하고 그 사용법을 익히도록 하였다. 이번 실습에서 데이터를 block단위로 tranfer하는데에는 크게 어려움이 없었다. 데이터를 블록단위로 한번에 저장하고 불러오는 방법을 사용함으로써 사용되는 state값과 code size를 훨씬 단축 할 수 있었다. 하지만 필자는 Stack을 이용하여 Full Ascending 방법으로 Data를 transfer하였는데, 처음 저장할 때 자꾸 r0의 값이 본인이 원하는대로 저장이 안되고 거꾸로 저장이 되는 것 이었다. 따라서 처음 저장할 때는 한 개의 Register 값만 저장한 뒤 그 다음부터 Stack에 차례대로 저장하였다. 그리고 처음 sp(stack pointer)가 0으로 초기화 되어있기 때문에 Stack을 이용하여 데이터들을 tranfer하기 위하여 sp를 지정된 메모리 주소 0x40000으로 이동시킨 뒤 문제를 구현하였다.

2번 문제에선 코드를 아무리 봐도 알고리즘상 문제는 없어보였는데 디버깅시 Register 에 올바른 값이 들어오지않고 쓰레기값이 계속 들어왔었다. ini file 에 sp 에 허용된 범위를 추가 하지않았기 때문이었고, Full Ascending 방법으로 짜서 이를 0x00000000 부터 범위를 추가해주니 완전하게 마무리할 수 있었다.

이번 과제를 수행하면서 Stack을 이용해 그 전의 값들을 저장하고 필요시 따로 빼올수 있어서 편리한 Stack의 중요성을 다시 한 번 느낀 것 같다. C언어에서 포인터를 이용해 구현했던 Call by Value 와 Call by Reference 의 차이와 비슷해 더욱 중요하게 느껴졌다.

* 1. 결론 (10%)

이번 실습시간에는 block data transfer에대해 정확히 이해하고, 구현하는 데에 목적이 있었다.block data transfer는 data transfer와 동일하게 데이터를 register에서 Memory로 저장하거나 메모리에서 레지스터로 불러오는 것과는 동일하지만, 이 부분을 여러 개의 데이터를 한번에 처리한다는 장점이 있다. 원래대로라면 STM명령어와 LDR명령어를 반복문을 사용하여 Memory에서 Register로 계속 반복하여 접근해서 데이터를 전송하기 때문에 block단위로 data에 접근하여 data를 전송하는 방법은 code size와 state를 줄일 수 있는 아주 효과적인 방법이다.

Stack을 이용하는 방법에는 총 4가지 방법이 있다. Full ascending, Full descending, empty ascending, empty descending이다. Ascending은 오름차순이라는 뜻으로 지정된 주소값을 기준으로 주소값이 커지면서 접근하는 방식이고, descending은 주소값이 작아지면서 접근하는 방식이다. 필자는 Full ascending방식으로 해당 문제를 구현하였고, 다음부터는 다른 방식 으로도 문제를 풀어보며 해당 구조를 정확히 이해할 필요를 느꼈다.

1. 참고문헌(2%)

강의자료

이준환 / 어셈블리 프로그램 설계 및 실습 / 광운대학교 (컴퓨터공학과)/ 2017년