어셈블리 프로그래밍 설계 및 실습

실험제목: Report Sample

실험일자:2017년 9월 14일 (목)

제출일자:2017년 9월 20일 (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%)

A.제목

Data Transfer to / from Memory

B.목적

기본 명령어를 사용해 기본적인 어셈블리어 프로그래밍을 이해하고, Resister 의 값을 원하는 메모리에 저장하고, 원하는 메모리의 값을 register로 불러오는 연습을 한다.

2. 설계 (Design) (50%)

Problem. 1.

A.Pseudo code

|  |
| --- |
| {  r1=5, r2=10, r3=15;  load r0 = &TEMPADDR1;    store r1 = r0+1byte;  store r2 = r1+1byte;  store r3 = r2+1byte;  load r1 = r0+1byte;  load r2 = r1+1byte;  load r3 = r2+1byte;  if(r1>10)  r5 = 1;  else(r1<10)  r5 = 2;  else  r5= 3;  } |

B.Flow chart 작성

|  |
| --- |
|  |
| 위 그림은 pseudo code를 flow chart로 나타낸 그림이다. |

C.Result

|  |  |  |
| --- | --- | --- |
|  | |  |
| 1) 위의 왼쪽 그림은 처음 디버깅 했을 때의 화면이다. register값들이 모두 0으로 초기화 되어있음을 확인할 수 있다.  위 오른쪽 그림은 r0에는 주소값 r1, r2, r3에는 5, 10, 15를 저장한 후 register화면이다. | | |
|  |  | |
| 2) 왼쪽 그림은 r1, r2 ,r3의 값을 r0의 주소값을 기준으로 1, 2, 3바이트 뒤의 주소값에 저장한 후의 register화면이다. R0에 저장 되 있는 주소값은 바뀌지 않는 방법으로 저장하였기 때문에 R0에 저장되어있는 값은 바뀌지 않았고 위의 그림을 보면 r1, r2, r3의값이 해당하는 주소값에 저장이 되어있음을 확인할 수 있다. | |
|  |  | |
| 3) 왼쪽 그림은 r0의 저장되어있는 주소값의 기준으로 1,2,3바이트 뒤의 주소값에 저장되어있는 값을 r1, r2, r3에 불러온 후의 register화면이다. 해당하는 주소값에 현재 r1, r2, r3에있는값들과 같은 값들이 저장되어 있었으므로 r1, r2, r3에 저장되어있는 값이 변하지 않았음을 확인할 수 있다. | |
|  | | |
| 4) 위의 왼쪽 그림은 CMP r1, #10을한뒤에 해당하는 조건문을 디버깅한 후의 register화면이다. r1은 5이기 때문에 비교값인 10보다 작다. 따라서 MOVL0 r5, #2만 만족하기 때문에, r5의 값이 2로 바뀐 것을 확인할 수 있다.  5) 위의 가운데 그림은 CMP r2, #10을 한 뒤에 해당하는 조건문을 디버깅한 후의 register화면이다. r2는 10이기 때문에 비교값인 10과 같다. 따라서 MOVEQ r5, #3만  만족하기 때문에, r5의 값이 3으로 바뀐 것을 확인할 수 있다.  6) 위의 오른쪽 그림은 CMP r3, #10을 한 뒤에 해당하는 조건문을 디버깅한 후의 register화면이다. r3는 15이기 때문에 비교값인 10보다 크다. 따라서 MOVHI r5, #10만 만족하기 때문에, r5의 값이 1로 바뀐 것을 확인할 수 있다. | | |

D.Performance

|  |  |
| --- | --- |
|  |  |
| 위 그림은 해당 코드를 끝까지 debug한 Command화면과 Register화면이다. code size는 60 bytes, States는 26 이다. 따라서 Score는 60 \* 262 = 40,560 이다. | |

Problem. 2.

1. Pseudo code

|  |
| --- |
| {  R0 =1, r1=2, r3=3, r4=4;  load r2 = &TEMPADDR1;  store r0 = r2 + 1byte;  store r1 = r0 + 1byte;  store r3 = r1 + 1byte;  store r4 = r3 + 1byte;  load r5[32bits] = r2;  store r4 = r2;  store r3 = r2+1byte;  store r1 = r3+1byte;  store r0 = r1+1byte;  load r6[32bits] = r2; |

그 후r2주소값에있는 32bit의 메모리 공간에 있는 데이터를 r6에 저장하였다.

B.Flow chart 작성

|  |
| --- |
|  |
| 위 그림은 해당코드를 flow chart로 나타낸 그림이다. |

C.Result

|  |  |
| --- | --- |
|  |  |
| 1) 위의 왼쪽 그림은 처음 디버깅 했을 때의 화면이다. register값들이 모두 0으로  초기화 되어있음을 확인할 수 있다.  위 오른쪽 그림은 r0, r1, r3, r4에 1, 2, 3, 4를 저장하고 r2에는 주소값을 저장한 후 register화면이다. | |
|  |  |
| 2) 왼쪽 그림은 r2의 주소값 0x00004000기준으로 1바이트단위로 r0, r1, r3, r4의 값인 1,2,3,4 저장한뒤에 r2주소값에 32bit의 메모리공간에있는 데이터를 r5에 저장한 후의register화면이다. 기준이 되는 r2의주소값은 변경하지 않았고, 현재, little-endian 방식인 하위바이트의 값이 메모리상에 먼저 저장되는 방식으로 되어있기 때문에 r5에 0x04030201이 저장되어있음을 확인할 수 있다. 위의 그림은 r2의 주소값에 1바이트씩더하여 r0, r1, r3, r4값을 저장한 Memory address 화면이다. |
|  |  |
| 2) 왼쪽 그림은 r2의 주소값 0x00004000기준으로 1바이트단위로 r4, r3, r1, r0의 값인 4, 3, 2, 1 저장한 뒤에 r2주소값에 32bit의 메모리 공간에 있는 데이터를 r6에 저장한 후의 register화면이다. 기준이 되는 r2의주소값은 변경하지 않았고 , 현재, little-endian 방식인 하위바이트의 값이 메모리상에 먼저 저장되는 방식으로 되어있기 때문에 r6에 0x01020304가 저장되있음을 확인할 수 있다. 위의 그림은 r2의 주소값에 1바이트씩더하여 r4, r3, r1, r0값을 저장한 Memory address 화면이다. |

D.Performance

|  |  |
| --- | --- |
|  |  |
| 위 그림은 해당코드를 끝까지debug한Command화면과Register화면이다. code size는 60 bytes, States는 30 이다. 따라서 Score는 60 \* 302 = 54,000 이다. | |

3. 고찰 및 결론

A.고찰 (35%)

처음으로 어셈블리어를 상용해 실습해 봤던 시간이라 친숙하게 사용하지 못했던 것 같다. C 언어에서는 변수에 값을 저장할 때, x=0 같은 식을 이용해 값을 저장하는 것과 다르게 어셈블리 언어는 MOV 라는 명령어를 사용해 register에 값을 저장하기 때문에 C 언어보다 사용해야할 명령어가 많아 처음 배우는 단계 에서 복잡했던 것 같다.

bit 와 byte 의 차이점(1byte = 8bits) 을 정확하게 알고나서야 STRB 와 STR 의 차이를 확실히 이해 할 수 있었다.

메모리에 직접 접근하여 메모리 값을 변경시키는 것이 허용되지 않아서 Load 나 Store 명령어를 사용해서 메모리에 접근해야 하고, 값을 저장할 때는 register 를 사용해 메모리에 접근 해야 함을 확인할 수 있었다.

또한 메모리 저장방식에는 크게 big-endian 방식과 little-endian 방식 으로 두 가지가 있는데 big- endian 은 큰 단위가 앞에 나오고 little-endian 은 작은 단위가 앞에 나오게 하는 방식이다. 따라서 어셈블리 프로그램 설계시big-endian 은 작은 수부터 저장을하고, little-endian은 큰 수부터 저장을 해서 메모리 상에는 big-endian 은 큰 단위가 앞에 나오고, little-endian은 작은 단이가 앞에 나오는 것을 확인할 수 있었다.

B.결론 (10%)

이번 과제를 통해, 기본 명령어들의 사용법과 register 와 memory 사이의 데이터 이동에 대해 이해 할 수 있었고 memory에 저장된 값을 확인하기 위하여 ini파일로 범위를 지정하는 방법을 터득했다.

그리고 기본 명령어 사이 STR 과 LDR 의 사용이 아주 다양하고, 정확히 이해해야 한다는 점을 깨닳았다.

little-endian 방식과 big-endian 방식의 차이점을 알고 난 후 어셈블리 프로그램에 한 발자국 더 다가간 것 같다.

4. 참고문헌 (2%)

강의자료

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