어셈블리 프로그래밍 설계 및 실습

실험제목: Second Operand & Multiplication

실험일자: 2017년 09월 28일 (목)

제출일자: 2017년 10월 11일 (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

Second Operand & Multiplication

* 1. 목적

Multiplication operation 과 ADD, Shift 연산을 이용한 Second operand를 사용하여 코드를 구현한다. 그 후 Multiplication operation 과 Second operand 를 사용한 코드의 성능 차이를 비교해본다.

1. 설계 (Design) (50%)
2. Problem. 1.
   1. Pseudo code

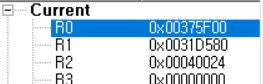
|  |
| --- |
| {  R0=1 // 1  Store r0 in r2, r2+=4byte  R0=R0+R0 // 1\*2  Store r0 in r2, r2+=4byte  R0=R0+R2\*2 // 1\*2\*3  Store r0 in r2, r2+=4byte  R0=R0\*4 // 1\*2\*3\*4  Store r0 in r2, r2+=4byte  R0=R0+R0\*4=R0\*5 // 1\*2\*3\*4\*5  Store r0 in r2, r2+=4byte  R1=R0+R0\*4  R0=R0+R1 // 1\*2\*3\*4\*5\*6  Store r0 in r2, r2+=4byte  R0=R0\*8-R0=R0\*7 // 1\*2\*3\*4\*5\*6\*7  Store r0 in r2, r2+=4byte  R0=R0\*8 // 1\*2\*3\*4\*5\*6\*7\*8  Store r0 in r2, r2+=4byte  R0=R0+R0\*8=R0\*9 // 1\*2\*3\*4\*5\*6\*7\*8\*9  Store r0 in r2, r2+=4byte  R1=R0+R0\*8  R0=R0+R1 // 1\*2\*3\*4\*5\*6\*7\*8\*9\*10  Store r0 in r2  } |

* 1. Flow chart 작성

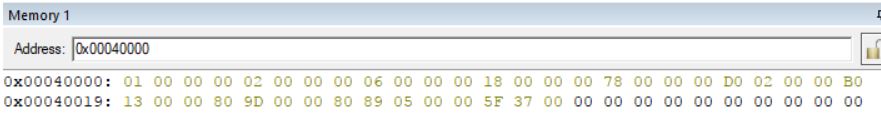
|  |
| --- |
|  |

위 그림은 Problem. 1. 의 Flow chart 이다.

* 1. Result



R0 에는 연산 후 값, R2 에는 주소값을 저장한다. Second operand 방법으로 순차적으로 구현하면 R0 에 10! , 16진수로는 375F00 이 R0에 정확하게 들어있음을 확인 할 수 있다.



Memory를 확인하면 주소 40000부터 시작하여 각 연산이 진행될 때 마다 4byte씩 증가하였고, 10! 의 16진수인 375F00 이 정확하게 저장되어있음을 확인할 수 있다. 이때 Little endian 방식 이므로 마지막 4byte를 확인할 때, 오른쪽에서 왼쪽으로 읽어서 확인해야한다.

* 1. Performance

|  |  |
| --- | --- |
| C:\Users\장한별 (Stanley)\Desktop\p1.code.JPG | C:\Users\장한별 (Stanley)\Desktop\p1.state.JPG |

위 그림을 보면 code size는 100bytes, State 는 35임을 확인할 수 있다.

따라서, Score = 100\*352 = 122,500 이다.

1. Problem. 2.
2. Pseudo code

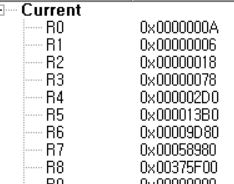
|  |
| --- |
| {  R1=1, R2=2, R3=3, R4=4, R5=5, R6=6, R7=7, R8=8, R9=9, R0=10  Store r1 in r10, r10+=4byte // 1  Store r2 in r10, r10+=4byte // 1\*2  R1=R2\*R3 // 1\*2\*3  Store r1 in r10, r10+=4byte  R2=R1\*R4 // 1\*2\*3\*4  Store r2 in r10, r10+=4byte  R3=R2\*R5 // 1\*2\*3\*4\*5  Store R3 in r10, r10+=4byte  R4=R3\*R6 // 1\*2\*3\*4\*5\*6  Store R4 in r10, r10+=4byte  R5=R4\*R7 // 1\*2\*3\*4\*5\*6\*7  Store R5 in r10, r10+=4byte  R6=R5\*R8 // 1\*2\*3\*4\*5\*6\*7\*8  Store R6 in R10, r10+=4byte  R7=R6\*R9 // 1\*2\*3\*4\*5\*6\*7\*8\*9  Store R7 in r10, r10+=4byte  R8=R7\*R0 // 1\*2\*3\*4\*5\*6\*7\*8\*9\*10  Store R8 in r10, r10+=4byte  } |

1. Flow chart 작성

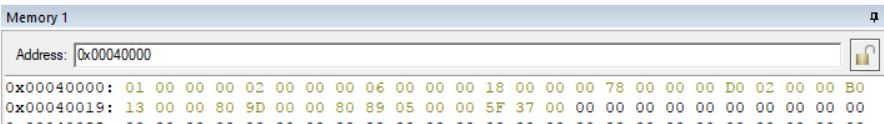
|  |
| --- |
|  |

위 그림은 Problem. 2. 의 Flow chart 이다.

1. Result



위 그림을 확인하면 R0 에 1\*2\*3 즉 3! 을 저장, 그 후 4를 MUL 하여 R1 에 저장했음을 확인할 수 있다. 이런식으로 Register에 저장되어있는 값들을 바로 전 결과값이랑 MUL하여 10!을 구한다. 10! 의 16진수인 375F00 이 R8에 정확하게 저장되어있음을 확인할 수 있다.



Memory를 확인하면 40000 주소값부터 시작하여 순차적으로 4byte 마다 저장되었고 10!의 16진수인 375F00 이 정확하게 저장되어있음을 확인할 수 있다.

1. Performance

|  |  |
| --- | --- |
| C:\Users\장한별 (Stanley)\Desktop\p2.command.JPG | C:\Users\장한별 (Stanley)\Desktop\p2.state.JPG |

위 그림을 보면 Code Size는 124 bytes, States 는 49 인 것을 확인할 수 있다. 따라서 Score = 124\*492 = 297,724 이다.

1. 고찰 및 결론
   1. 고찰 (35%)

이번 실습을 통해 Multiplication operation 방식과 Second operand 방식을 이용해 1부터 10까지의 곱인 10! 을 구현했다. Problem. 1. 같은 경우 ADD 와 Shift 연산을 이용해 Second operand 방식, Problem. 2. 에서는 Multiplication operation 방식을 사용했다. 먼저 Second operand 방식으로 구현할 때 까다로웠던 점은 6! 과 10! 을 구할 때는 한 줄로 표현을 하지 못했다. 6과 10을 곱하는 부분은 그 전 값들을 Register 에 따로 저장하지않았기 때문에 두 줄로 표현을 했다.

두번째로 Multiplication operation 방식으로 10! 을 구할 때는 크게 문제 없이 구현했다. 1부터 10까지 곱하는 연산을 그 전 결과값에 곱할 인자값을 1씩만 올려준 후 MUL하면 간단히 해결됐다. 하지만 Register의 개수가 많아지는 문제점이 있었다.

Problem. 1. 의 Codesize 는 100 bytes, state 는 35, Problem. 2. 의 Codesize는 124bytes, state 는 49였다. 확실히 Multiplication operation 방식이 Second operand 방식보다 Codesize 도 크고 State 도 크다.

따라서 Second operand를 사용하는 이유는 state 수와 사용되는 bytes수를 줄이기 위해 사용한다. 곱셈을 할 때 MUL 연산을 사용하면 한 번에 많은 byte(2byte)를 차지한다. 이때 Register 에 값들을 다 할당을 해준 뒤 연산을 해야하기때문이다. 따라서 Second operand 방식으로 곱 연산을 1byte로 표현한다면, 사용되는 Codesize도 줄어들고 사용되는 Register의 개수도 줄일 수 있다.

* 1. 결론 (10%)

Second operand 방식과 Multiplication operation 방식의 성능을 비교해본 결과 Second operand 방식이 Multiplication operation 방식보다 Codesize도 작고, state 도 더 작으므로 Second operand 방식이 성능이 뛰어나다고 할 수 있다.

하지만 Second operand 방법을 사용하면 한눈에 그 의미를 알아차리기 힘들 수 도 있다. Multiplication operation 방식은 훨씬 간결하기 때문에 이해하기가 쉽고, 구현방법이 간단하다. 각각의 방식에는 장단점이 있기 때문에 사용자가 편한 방식을 이용해 사용하면 될 것 같다.

1. 참고문헌(2%)

강의자료

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