어셈블리 프로그래밍 설계 및 실습

보고서

실험제목: Second Operand & Multiplication

실험일자: 2018년 10월 04일 (목)

제출일자: 2018년 10월 11일 (목)

학 과: 컴퓨터공학과

담당교수: 이준환 교수님

실습분반: 화 5, 수 6,7

학 번: 2015722025

성 명: 정용훈

1. 제목 및 목적

A. 제목

Second Operand & Multiplication

B. 목적

Multiplication operation을 사용하여 코드를 작성하는 것과 Second operand를 사용하여 코드를 작성하는 것에 대한 code size와 state를 비교한다. 또한 Operand의 순서에 따른 성능의 차이에 대하여 알아볼 수 있다. 최종적으로는 어떤 식으로 설계를 해야 가장 효율적인 프로그램이 되는지 생각해볼 수 있다.

2. 설계 (Design)

A. Pseudo code

Problem1

Shift란 뜻은 해당 숫자를 이진수로 만들었을 때 각 숫자들을 이동시킨 횟수이다. Ex) 왼쪽으로 1 shift: 0011->0110

(1) R0레지스터에 Memory의 주소 값을 저장한다.

(2) R1레지스터에 1의 값을 저장한다.

(3) R2레지스터에 1을 왼쪽으로 1 shift한 값을 저장한다. (r2=1*2)

(4) R3레지스터에 r2의 값 더하기 r2의 값을 왼쪽으로 1 shift한 값을 저장한다. (r3=r2*3)

(5) R4레지스터에 r3의 값을 2 shift한 값을 저장한다. (r4=r3*4)

(6) R5레지스터에 r4의 값 더하기 r4의 값을 왼쪽으로 2 shift한 값을 저장한다. (r5=r4*5)

(7) R6레지스터에 r5의 값을 2 Shift한 값에서 r5의 값을 뺀다. 그 후 r6에 r6의 값을 왼쪽으로 1 shift한 값을 저장한다. (r6=r5*6)

(8) R7레지스터에 r6의 값을 왼쪽으로 3 shift한 값에서 r6의 값을 빼고 저장한다. (r7=r6*7)

(9) R8레지스터에 r7의 값을 왼쪽으로 3 shift한 값을 저장한다. (r8=r7*8)

(10) R9레지스터에 r8의 값 더하기 r8의 값을 왼쪽으로 3 shift한 값을 저장한다. (r9=r8*9)

(11) R10레지스터에 r9의 값 더하기 r9의 값을 왼쪽으로 3 shift한 값을 저장하고, r10레지스터에 r10의 값 더하기 r9의 값을 저장한다. (r10=r9*10)

(12) R10의 값을 메모리에 저장 후 프로그램을 종료한다.

Problem2

(1) R0레지스터에 Memory의 주소 값을 저장한다.

(2) R1레지스터에 1의 값을 저장한다.

(3) R11에 2의 값을 저장한다.

(4) 레지스터R2에 r1*r11의 값을 저장한다. r11에 1을 더한다. (r2=1*2)

(5) 레지스터R3에 r2*r11의 값을 저장한다. r11에 1을 더한다. (r3=r2*3)

(6) 레지스터R4에 r3*r11의 값을 저장한다. r11에 1을 더한다. (r4=r3*4)

(7) 레지스터R5에 r4*r11의 값을 저장한다. r11에 1을 더한다. (r5=r4*5)

(8) 레지스터R6에 r5*r11의 값을 저장한다. r11에 1을 더한다. (r6=r5*6)

(9) 레지스터R7에 r6*r11의 값을 저장한다. r11에 1을 더한다. (r7=r6*7)

(10) 레지스터R8에 r7*r11의 값을 저장한다. r11에 1을 더한다. (r8=r7*8)

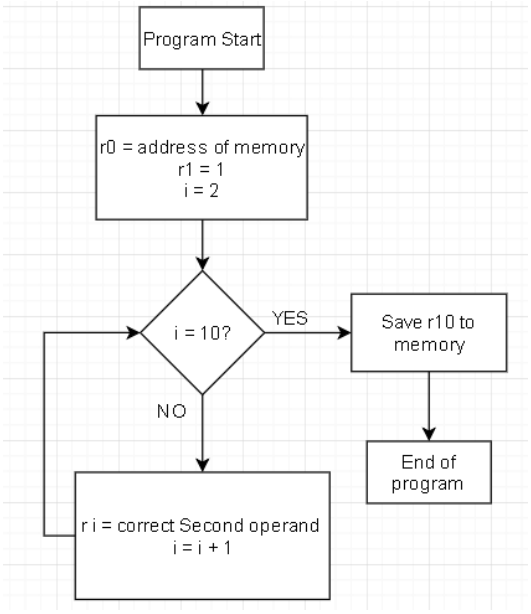
(11) 레지스터R9에 r8*r11의 값을 저장한다. r11에 1을 더한다. (r9=r8*9)

(12) 레지스터R10에 r9*r11의 값을 저장한다. (r10=r9*10)

(13) R10의 값을 메모리에 저장 후 프로그램을 종료한다.

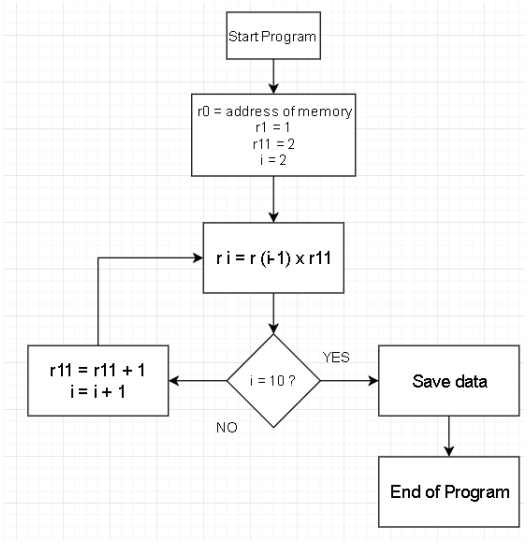
B. Flow chart 작성

Problem1



Problem 1에 관련한 Flow chart입니다. Second operand는 일정한 규칙이 있는 것이 아니라 구현하고자 하는 곱연산에 따라 방법이 여러가지가 나올 수 있기 때문에 해당 flow chart에서는 "알맞은 식을 넣는다"로 대체하여 썼습니다. 코드와 마찬가지로 2번째 Problem보다 간단하게 chart를 완성할 수 있습니다.

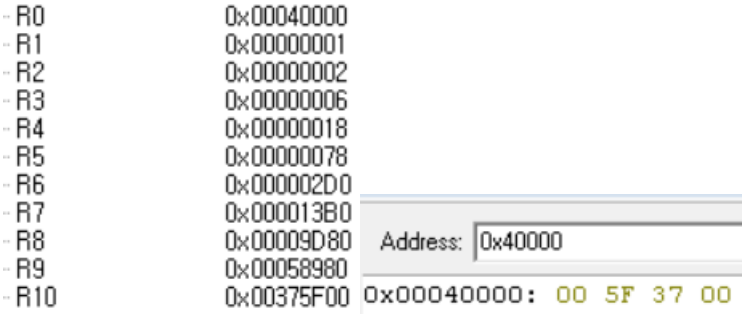
Problem2



R11에 값을 하나씩 추가해 주면서 곱하는 수를 일정하게 늘려서 다음 수와 계속 곱하는 chart입니다. Chart를 보면 Factorial의 기능을 하는 것을 볼 수 있습니다.

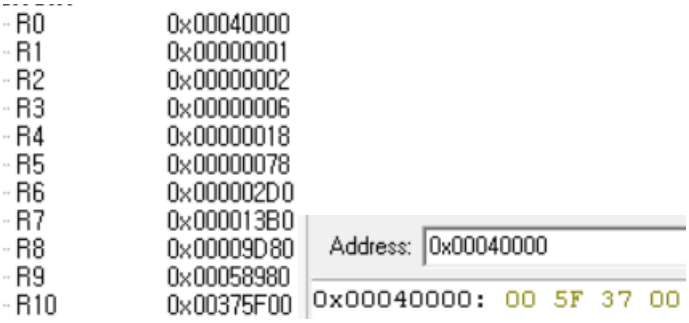
C.Result

Problem1



다음 실행 결과는 Second Operand를 사용하여 나타낸 결과 값입니다. 레지스터 R0에는 가져올 메모리의 정보를 넣어주고 R1에는 1의 값을 저장해주고 ADD, LSL, MOVE, RSB 등을 이용하여 Factorial을 구현한 모습입니다. 해당 레지스터들의 값을 점화 식으로 간단하게 나타내면 다음과 같이 나타낼 수 있습니다. $R_n=R_{n-1}*n$ (단, 1<n<11, n은 자연수)

Problem2



다음 실행 결과는 multiplication을 사용한 결과 값입니다. 결과 자체에는 첫번째 문제와 다른 점이 없습니다. 대신 두 실습의 차이로는 Code의 사이즈와 state다르게 나타납니다. 두 실습의 차이점은 다음 항목에서 자세하게 설명하도록 하겠습니다.

D. Performance

Problem1

```
Program Size: Code=64          States          17
```

해당 항목을 지금까지는 사용하지 않았지만 code사이즈와 state에 관련한 설명이 더 필요하다고 판단되어 작성하게 되었습니다. 위에 보이는 code size와 state는 첫 번째 실습 즉, Second operand를 사용하여 작성한 코드의 size와 state입니다. Problem2에서 보이는 size와 state에 비에 적은 것을 확인 할 수 있습니다. 이처럼 곱 연산 대신에 shift연산을 사용하면 프로그램을 좀더 효율적으로 작성할 수 있다는 것을 알 수 있습니다.

Problem2

```
Program Size: Code=92          States          33
```

다음은 Problem2의 code size와 state입니다. Probelm1에 비해 비교적 숫자가 크게 나온 것을 확인 할 수 있습니다. 그 이유로는 곱연산을 하는 원리에서 비롯됩니다. 곱 연산을 하기위해서는 비트 수에 따른 여러 개의 adder가 필요한데 add와 shift를 이용하는 첫번째 방법에 비해 add를 여러 번 반복해서하기 때문에 첫번째 실습에 비해 코드의 효율성이 떨어집니다.

3. 고찰 및 결론

A. 고찰

해당 실습을 통해서 Factorial을 여러가지 방법으로 구현하는 방법을 알게 되었습니다. 또한 곱연산기를 사용하는것과 사용하지 않는 것에 대한 프로그램의 큰 차이도 알게 되었습니다. 또한 첫번째 실습에서 코드를 작성하던 중 6을 곱하는 연산과 10을 곱하는 연산이 2줄을 거쳐 연산 되는 것을 보고 코드의 사이즈를 줄이기 위해서 한 줄로 작성하는 방법은 없을지 계속 고민하게 되었습니다. 최종적인 제 능력으로는 해당 곱을 구현하기 위해서는 두줄의 코드를 사용하는 것이 맞다고 생각하여 실습에서는 코드 두줄을 통해 6연산과 10연산을 구현하게 되었습니다. 다음으로는 두번째 실습을 진행하면서 곱연산을 사용하는데 최대한 레지스터를 적게 사용하고 싶어 값을 저장하고 있는 레지스터에 곱연산을 통해 값을 업데이트 시켜 주기 위하여 다음과 같은 식을 사용했습니다. (mul r1, r1, r11) 하지만 해당 연산은 앞 두 인자에 같은 레지스터를 쓸 수 없게 되어있고 여러 개의 레지스터를 사용하여 Factorial을 구현하게 되었습니다.

B. 결론

실습을 하면서 평소 high level의 언어를 배우면서 아무렇지 않게 사용하였던 덧셈 연산이나 곱셈 연산이 기능의 차이와 처리속도가 많이 차이 난다는 것을 모르고 있었습니다. 직접 state와 code size를 보면서 차이를 느낄 수 있었습니다. 실습을 하면서 Second operand의 연산이 state또한 효율적이며 code size도 적은 것을 확인할 수 있었습니다. 하지만 첫번째 문제를 풀면서 무조건 Second operand를 사용한다고 다 좋은 것은 아니라고 생각됐습니다. 바로 6의 곱연산과 10의 곱연산을 구현하는데 느낀 점입니다. 6과 10의 경우 코드를 두줄에 구현하였지만 곱연산을 사용하는 경우 한 줄로 표현할 수도 있는 식입니다. 이렇게 때에 따라서 곱연산의 효율성이 더 커질 수 있는 프로그램이 없다고 생각하지 않기 때문에 프로그램을 구현하는데 한 방법만 쓰는 것이 아니라 여러가지 방법을 선택해서 프로그램을 작성하는 것이 가장 효율적으로 프로그램을 작성할 수 있다고 생각됩니다.

4. 참고문헌

이준환/ARM instruction set - second operand & multiplication/광운대학교/2018