어셈블리 프로그래밍 설계 및 실습

보고서

실험제목: Control flow & Data Processing

실험일자: 2018년 09월 20일 (목)

제출일자: 2018년 10월 04일 (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

Control flow & Data Processing

B. 목적

Memory로부터 값을 불러와 연산을 수행하며 다시 Memory에 저장하는 방법을 익히며 Branch명령어를 이해하고 사용하며 conditional execution과의 차이를 생각해보고 같은 동작을 하는 코드를 여러 방법으로 작성하여 코드의 performance를 생각해 볼 수 있다.

2. 설계 (Design)

A. Pseudo code

Problem 1

(1) 각 레지스터에 문자열과 메모리 주소 결과를 알리기 위한 값을 저장합니다.
R0 <<비교 대상 문자열 1, R1 비교 대상 문자열 2, R3 <<10, R4 <<11

(2) R0와 R1에 있는 문자열을 하나씩 가져와 문자를 비교합니다.

(3) 비교 결과가 같으면 4번 문항으로 가고 다르면 5번 문항으로 갑니다.

(4) 가져온 문자열을 0과 비교하였을 때 같으면 6번 다르면 다시 2번 문항으로 갑니다. (Loop를 결정하는 구간)

(5) R3의 정보를 R2에 저장 되어있는 메모리 주소로 옮겨주고 문항 7로 갑니다.

(6) R4의 정보를 R2에 저장 되어있는 메모리 주소로 옮겨주고 문항 7로 갑니다.

(7) 프로그램을 종료합니다.

Problem 2

(1) 각 레지스터에 값이 저장되어 있는 배열의 주소와 메모리 주소를 저장합니다.
R0 <<메모리 주소 저장, R1 <<값을 저장하고 있는 배열 저장(10,9,8,7,6,5,4,3,2,1)

(2) 현재 배열이 가리키고 있는 정보를 맨 뒤로 옮겨 준다.

(3) 뒤에 있는 정보를 메모리로 옮겨준다. 만약 해당 문항이 반복된 거라면 가리키는 정보를 하나 앞으로 옮긴 후 저장한다. 그 후 배열의 첫번째 요소와 비교하였을 때 다르면 3번을 반복하고 같으면 4번 문항으로 간다.

(4) 프로그램을 종료합니다.

Problem 3-1

(1) 각 레지스터에 사용할 정수를 저장하고 메모리 주소를 저장한다.
R0 <<메모리 주소 저장, R1 <<1 저장
(2) 1을 사용하여 11을 만들기 위하여 shift 연산과 덧셈을 한다.
(3) 비교 대상이 될 숫자 29를 만들기 위하여 R3의 값을 사용하여 shift연산 실행
(4) 순차적으로 R3의 값을 +2 해주면서 R5에 새로 값을 저장해준다.
(5) R3의 값을 29와 비교하였을 때 같으면 R5의 값을 메모리에 저장하고 프로그램을 종료하고 아니면 4번을 반복 수행한다.

Problem 3-2

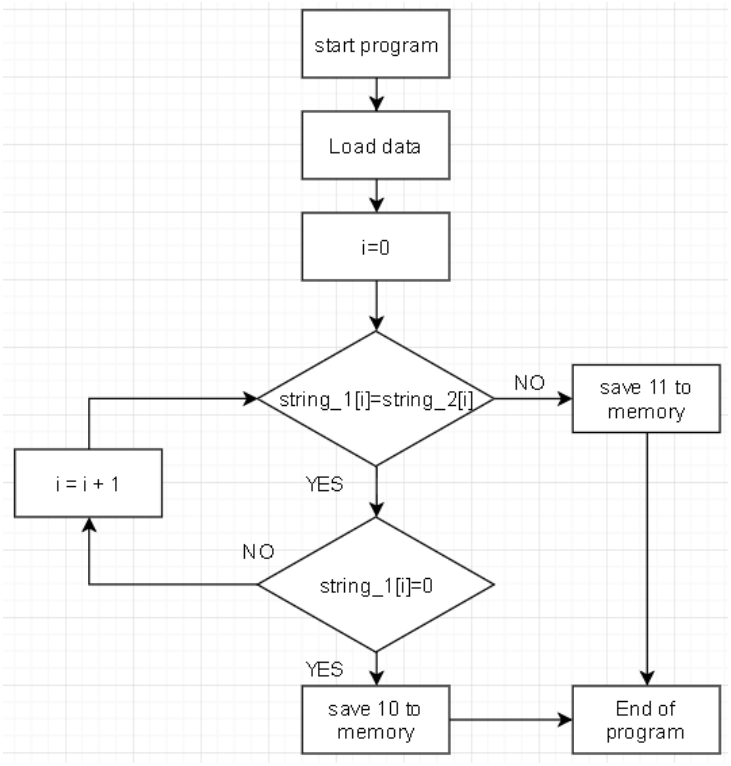
(1) 각 레지스터에 사용할 정수를 저장하고 메모리 주소를 저장한다.
R0 <<메모리 주소 저장, R1 <<1저장, R2 <<10저장
(2) R1의 정보를 가지고 Shift연산관 ADD연산으로 10을 만든다.
(3) R3=R2+R1
(4) R4=R3*R2, 연산이후 R4의 값을 R0 메모리에 저장하고 프로그램을 종료한다.

Problem 3-3

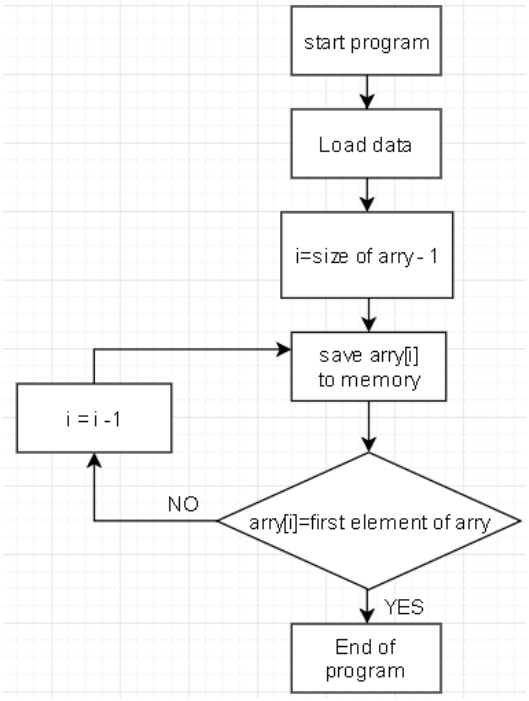
(1) 각 레지스터애 사용할 메모리 주소와 정보가 담긴 배열의 주소를 저장한다.
R0 <<메모리 주소 저장, R1 <<배열의 주소 저장(11,13,15,17,19,21,23,25,27,29)
(2) R3에 R1의 첫번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(3) R3에 R1의 두번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(4) R3에 R1의 세번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(5) R3에 R1의 네번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(6) R3에 R1의 다섯 번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(7) R3에 R1의 여섯 번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(8) R3에 R1의 일곱 번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(9) R3에 R1의 여덟 번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(10) R3에 R1의 아홉 번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(11) R3에 R1의 열 번째 원소 저장 후 다음 연산 실행 R2=R2+R3
(12) R2의 값을 메모리 R0에 저장 후 프로그램 종료

B. Flow chart 작성

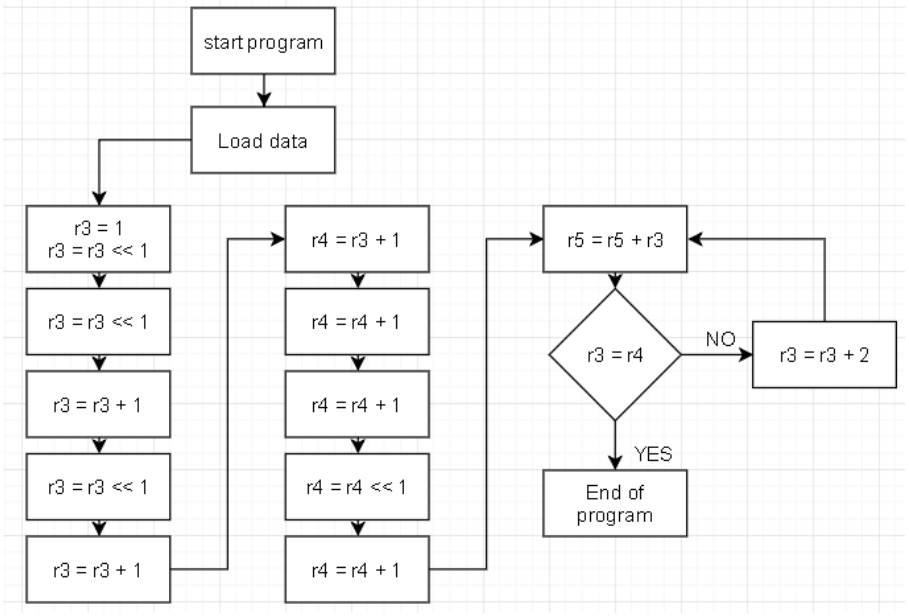
Problem 1



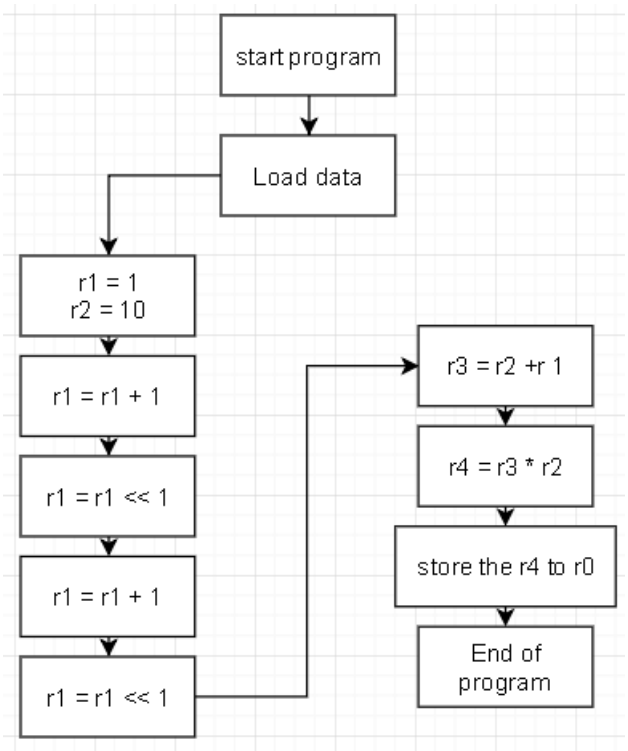
Problem 2



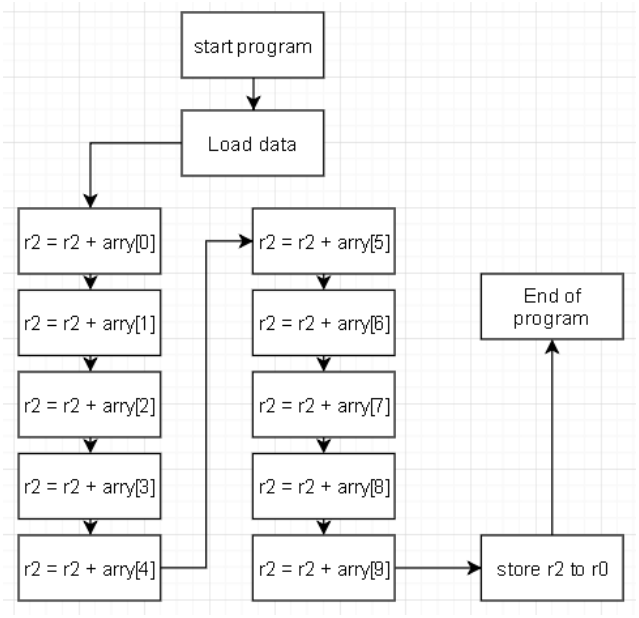
Problem 3-1



Problem 3-2



Problem 3-3


start program
Load data
r2 = r2 + arry[0]
r2 = r2 + arry[1]
r2 = r2 + arry[2]
r2 = r2 + arry[3]
r2 = r2 + arry[4]
r2 = r2 + arry[5]
r2 = r2 + arry[6]
r2 = r2 + arry[7]
r2 = r2 + arry[8]
r2 = r2 + arry[9]
End of program
store r2 to r0

C. Branch와 Conditional execution의 차이점과 성능 차이

해당 과제를 해결하는데 있어 필수적으로 사용한 요소가 Branch와 Conditional execution입니다. 두 요소의 차이를 쉽게 설명하자면 C++에서 우리가 호출하는 함수와 if, for ...등등 함수를 구성하는 내부에 쓰이는 조건들의 차이라고 생각하면 이해하기 쉽습니다. 두 요소는 상황의 따라 효율이 달라 질 수 있습니다. 간단한 동작을 구현하기 원한다면 단순히 Conditional execution을 사용하여 설계하는 것이 편하고 좀더 동작이 복잡하다면 Branch로 작성하여 따로 호출할 수 있도록 설계하는 것이 코드를 해석하는 입장에서 보았을 때 좀더 쉽게 이해할 수 있고 보기 깔끔할 수 있습니다. 아래 C++의 코드와 어셈블리의 코드를 비교하면 쉽게 이해할 수 있습니다.

```
else if (ll->pHead == NULL)
{
    while (!info.eof())
    {
        info >> carinfoN;
        info >> carinfoO;
        info >> carinfoS;
        if (strlen(carinfoO)<3)
            break;
        CarNode *newNode = new CarNode;

        newNode->SetcarNum(carinfoN);
        newNode->SetcarOwner(carinfoO);
        newNode->Setstate(carinfoS);
        ll->Insert(newNode);
    }
}
```

```
Loop
    ADD r5,r5,r3 ;r5=r5+r3
    CMP r3,r4 ;compare the r3 and r4
    BEQ Finish ;if equal, go to Finish
    ADD r3,r3,#1 ;r3=r3+2
    ADD r3,r3,#1
    BNE Loop ;if not equal, go to Loop
```

빨간색 네모는 보다 복잡한 알고리즘을 수행하기위한 Branch호출 즉 함수 호출이라고 할 수 있으며 초록색으로 표시된 부분은 조건에 따라 수행되는 코드 즉 Conditional execution입니다.

D. Result

Problem1

문자열이 같은 경우

```
Char1 DCB "aaa",0
Char2 DCB "aaa",0
```

```
0x00004000: 0A 00 00 00 00 00 00 00 00 00
```

이와 같이 메모리에 10의 값이 들어가 있는 것을 확인할 수 있다. 그에 반해 문자열이 다른 경우

```
Char1 DCB "aca",0
Char2 DCB "aaa",0
```

```
0x00004000: 0B 00 00 00 00 00 00 00 00 00
```

위 이미지와 같이 메모리에 11의 값이 저장되어 있는 것을 확인 할 수 있다.

```
R8          0x00000063
R9          0x00000061
```

옆 이미지는 문자열을 숫자로 변환하여 비교하는 과정이며 이와 같이 숫자가 다른 경우 조건문을 통하여 알맞은 함수를 실행한다.

Problem2

결과를 먼저 보자면 아래와 같은 이미지로 메모리의 값이 결정된다.

```
0x00004000: 01 02 03 04 05 06 07 08 09 0A
```

메모리에서 불러오는 값을 배열의 맨 뒤로 설정 해준 후 순차적으로 뒤에서부터 정보를 가져와 메모리에 넣으면 값이 반전되어 위와 같이 메모리에 값을 갖는다.

Problem3-1

11+13+15+.....+27+29의 연산 결과를 메모리에 저장하는 문제이다. 결과 값은 다음과 같이 나온다. `0x00004000: C8 00 00 00 00 00 00` 과정으로는 숫자 1만 사용하여 shift연산 과정을 거쳐서 만들어진 숫자로 Loop를 돌려 위와 같은 연산을 하여 메모리에 저장하는 것이 과제이다.

Problem3-2

3-2의 과제는 결과 값은 같지만 과정이 다르다. 결과부터 보자면 다음과 같이 나온다. `0x00004000: C8 00 00 00 00 00 00` 위 결과와 동일한 결과이며 과정으로는 흔히 사용하는 n*(n+10)의 연산을 하여 나온 결과를 메모리에 저장한 것이다.

Problem3-3

3-3 과제도 마찬가지로 결과 값은 같은 결과가 나온다. 이번 과제의 결과도 다음과 같다. `0x00004000: C8 00 00 00 00 00 00` 같은 값이 나오며 과정으로는 unrolling 방식을 사용하였다 unrolling이란 특별한 조건이나 loop를 사용하지 않고 있는 그대로 식을 풀어 쓴 방식을 말한다.

3. 고찰 및 결론

A. 고찰

해당 과제를 해결하면서 가장 큰 문제는 문자를 불러오는 방법에 있었습니다. 처음에는 문자를 비교하는데 있어 일반 C++이나 java에서처럼 문자를 한번에 비교하는 방법을 생각하여 문자를 저장하고 있는 레지스터 전체를 비교하는 방법으로 코드를 설계하였습니다. 하지만 같은 문자열을 비교할 때도 문자열이 다른 경우로 인식되는 것을 보았고 그 이유는 해당 문자열에 대한 값을 저장하는 것이 아니라 레지스터 자체는 문자가 저장되어 있는 메모리를 저장하고있다는 것을 알게 되었습니다. 그리하여 저장 되어있는 문자를 원소별로 한 문자 한 문자 비교하는 방법을 택하였고 코드를 설계하며 문제를 해결하게 되었습니다. 또한 3번의 각각의 문제를 해결하면서 branch와 loop, conditional execution의 중요성을 알게 되었습니다. 해당 실습에서는 unrolling방식을 통하여 code size와 state를 비교한 결과 이번 실습에서는 unrolling방식의 성능이 가장 효율적으로 나타났습니다. 하지만 개인적인 생각으로는 unrolling방식만 사용하는 것이 아니라 프로그램의 크기와 알고리즘에 따라 branch와 loop를 적절히 사용하면 충분히 성능을 좋게 작성할 수 있습니다.

B. 결론

특히 이번과제의 3번 문제를 해결하며 branch와 Loop의 중요함을 알게 되었습니다. 간단하게 말하면 프로그램은 어떠한 문제를 수행하는데 있어 그것을 해결해주는 역할을 합니다. 그 문제를 해결하기 위해서는 반복적인 패턴의 행동을 하게 됩니다. 이러한 반복적인 패턴을 수행하는데 있어 3-3번 문제였던 unrolling의 방식은 code size가 다소 크지만 state가 적고 단순히 이해하고 작성하기에는 편하지만 수행해야 할 일과 프로그램이 커질수록 (풀어야할 코드가 많을 수록) 비효율 적이고 코드가 뭉게 질 수 있다고 교수님께 답변을 들었습니다. 이러한 문제를 해결하기 위해서는 과제를 하면서 사용한 Loop나 branch를 적절히 같이 사용하여 특정 조건에 따라 수행하는 일을 반복적으로 할 수 있게 도와주거나 다른 함수를 호출하여 문제를 해결하는데 훨씬 효율적으로 수행할 수 있습니다. 이러한 방법을 응용하였을 때 어떠한 자료구조에서 정보를 찾는다고 하였을 때 Loop를 통하여 반복적으로 정보를 찾는 것을 수행할 수 있으며 branch를 통하여 정보를 찾았을 경우 그 정보를 가시적으로 표현하거나 다른 함수를 호출하여 다른 작업을 할 수 있는 상황을 생각해 볼 수 있습니다.

4.참고문헌

이준환/ARM instruction set - control flow & data processing/광운대학교/2018