어셈블리 프로그래밍 설계 및 실습
보고서

실험제목: Control flow & Data processing

실험일자: 2018년 09월 18일 (화)

제출일자: 2018년 09월 25일 (화)

학 과: 컴퓨터공학과

담당교수: 이형근 교수님

실습 분반: 화 6,7 , 수 5

학 번: 2017202010

성 명: 박유림

1. 제목 및 목적

A. 제목

Control flow & Data processing

B. 목적

저번에 배운 memory로부터 값을 받아 이를 이용해 원하는 연산을 수행할 수 있고 memory를 할당하여 data를 저장한다.

이번 수업시간에 배운 곱셈 연산자를 이용하여 원하는 작동을 구현할 수 있다. 또한 performance를 생각하며 state등을 비교하여 생각한다. 마지막으로 branch 명령어를 통해 구현을 원하는 코드를 작성한다.

2. 설계 (Design)

A. Pseudo code

1. Problem1

```
r0[5]="Hello"
r1[5]="Helle"
r8=5
for(int i=0; i<r8; i++){
   if(r0[i]==r1[i]) continue
   else r3 = 0x0B
}
r3 = 0x0A
```

memory의 0x00004000 번지에 레지스터 r3안에 있는 값을 넣어준다.

2. Problem2

```
r0[10] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
r6=10
for(int r5=0; r5<r6; r5++){
      mem[0x00004009-i] = r0[i]
}
```

배열인 r0안의 값들을 거꾸로 memory의 지정한 곳에 채워준다.

3. Problem3_1

```
r0 = 1
r1 = r0 << 1 // it's 2
r2 = r0 << 3 // it's 8
r3 = r0 + r2 + r1 // it's 11
r4 = r3

r7 = 9 // it's indexing number

for(int r6=0; r6<r7; r6++){
  r3 = r3 + r1
  r4 = r3 + r4
}
```

4. Problem3_2

```
r0 = 1
r1 = r0 << 1
r2 = r0 << 3

r3 = r2 + r1 // it's 10
r4 = r3 + r3 // it's 20

r5 = r4 * r3 //because of sum of equal difference progression
```

등차수열의 합은 아래와 같다.

$$\frac{2a_1 + (n-1)d}{2}n$$

(이때, a_1은 초항, d는 등차의 값, n은 항의 갯수)

이 식에 맞춰서 값들을 대입해보면 11~29 등차수열의 합의 공식은 n(n+10)이고, 항의 개수는 10개이다. 따라서 10*20=200이 된다.

5. Problem3_3

r0 = 11

r1 = 13

r2 = 15

r3 = 17

r4 = 19

r5 = 21

r6 = 23

r7 = 25

r8 = 27

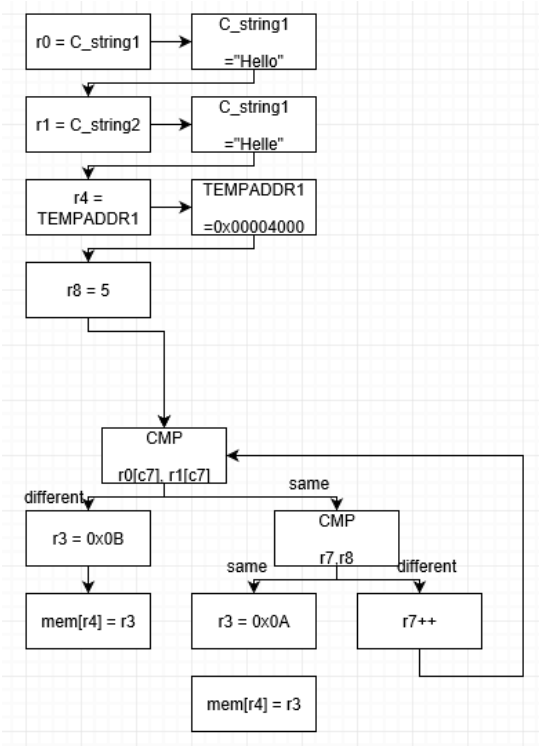
r9 = 29

r10 = r0 + r1 + r2 + r3 + r4 + r5 + r6 + r7 + r8 + r9

레지스터에 등차 수열의 모든 항들을 r0 ~ r9까지 모두 넣어준다. 그 이후 r10에 모든 레지스터의 값들을 더한다.

B. Flow chart 작성

1. Problem1

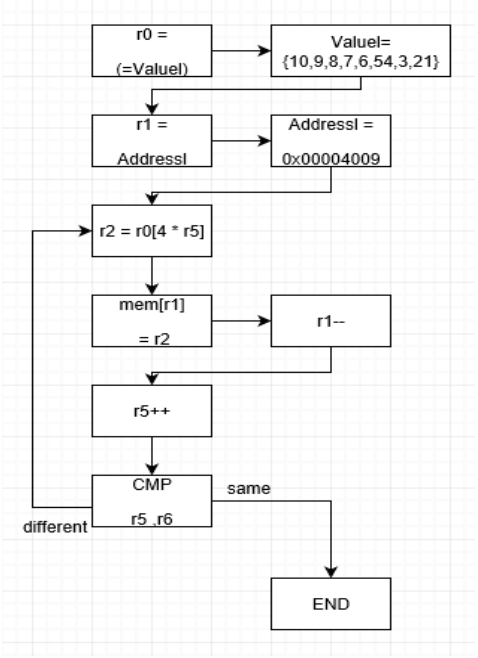


처음으로 비교할 string을 두개 만들어주어야 한다. 이 string의 이름을 각각 C_string1, C_string2라고 하자. C_string1에 "Hello"를 넣어준 후 r0에는 C_string1의 주소 값을 넣어준다. 마찬가지로 C_string2에 "Helle"를 넣어준 후 r1에 C_string2의 주소 값을 넣어주었다. 그 후, r4에 TEMPADDR1을 넣어주는데 이때, TEMPADDR1은 0x00004000 즉 결과값이 들어갈 메모리 주소가 할당된다.

그 후에 loop를 돌 때, loop를 돌 횟수를 정하는 숫자를 r8에 대입해준다. 본격적으로 loop를 돌기 위해서, r0의 r7번째 값과 r1의 r7번째 값을 비교해서 다르면 r3에 0x0B를 대입한다. 같은 경우에는 r7과 r8을 비교하여 같으면 r3를 0x0A에 대입하고, 다를 경우에는 r7의 값을 하나 증가시켜 다음 r0와 r1를 비교하는 Loop를 돈다.

마지막으로 r3에 저장된 값을 r4에 저장된 주소값에 대응하는 memory에 넣어준다.

2. Problem2

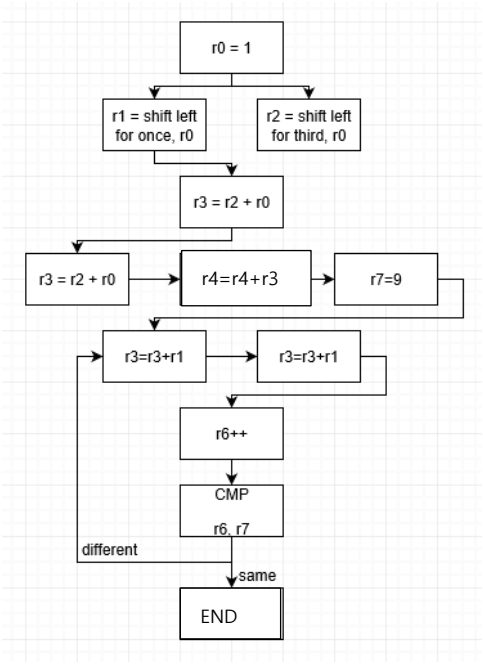


Valuel이라는 배열에 10~1의 값들을 배열의 원소들로 넣어준 후 Valuel의 주소값을 r0에 대입해준다. Addressl에 0x00004009라는 주소 값을 넣고 r1에 그 변수를 할당하여 주소 값을 전달한다. r2에 r0의 n번째 값을 넣어준다. 이때, n번째 값은 4*r5(숫자의 크기가 4이고 r5는 indexing number이다.)이다. r1에 저장된 주소 값에 r2의 data를 저장하고 r1을 하나 뺀다. 그

이후 indexing number인 r5를 하나 늘려 준다.

Indexing number(r5)와 array's size(r6)의 크기를 비교한 후, 같으면 프로그램을 끝내고, 다르면 Loop로 다시 돌아간다.

3. Problem3

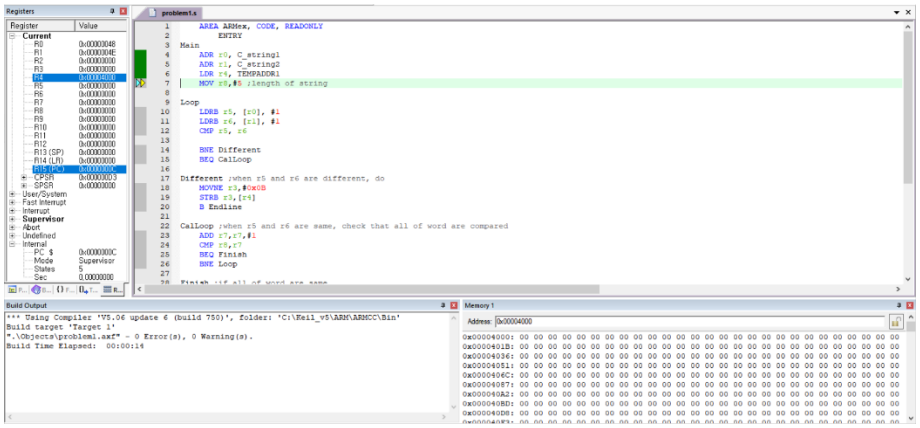


Shift와 1을 사용해 11을 만들어야 한다. 11=1+2+8 임을 유의해서 r0에 1을 넣는다. r1에 r0를 왼쪽으로 한 칸 shift한 값을 , r2에는 r0를 왼쪽으로 세칸 shift한 값을 대입하고 이 세 레지스터를 더해서 r3에 넣으면, 그 값이 11이 된다.

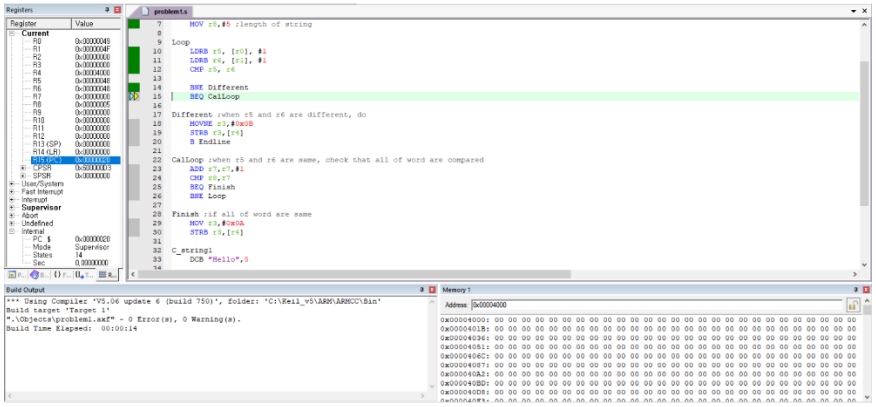
Loop를 돌기 위해 indexing number로 r7=9를 저장한다. 본격적으로 Loop안으로 들어가서, r3=r3+r1으로 다음 등차수열의 원소를 r3로 지정한 후, 등차수열의 합을 나타내는 r4에 r3를 더해준다. 다음으로 indexing number(r6)를 하나 올리고, size를 나타내는 r7과 비교한다. 다르다면 다시 Loop의 시작으로 가고 같으면 END시킨다.

C. Result

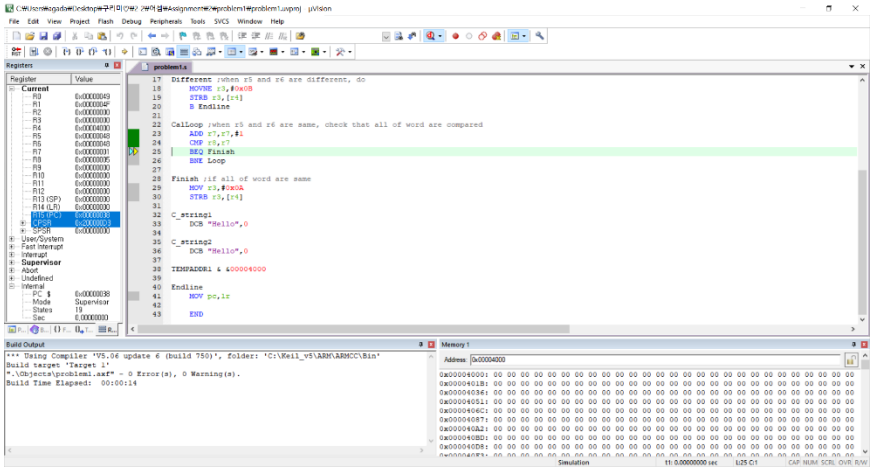
1. Problem1



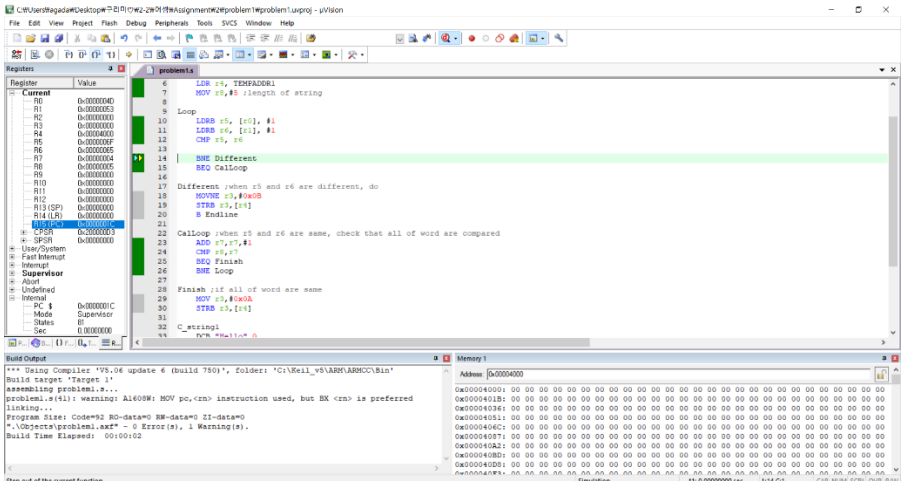
Main 부분만을 처음 실행한 경우, r0에는 C_string1의 문자열 주소 값이 들어가고, r1에 C_string2 문자열 주소 값이 대입된다. r4에는 아래 선언해준 TEMPADDR1이란 이름을 붙인 주소 값, 0x00004000이, 마지막으로 r8에는 반복문의 횟수인 5가 저장된다.



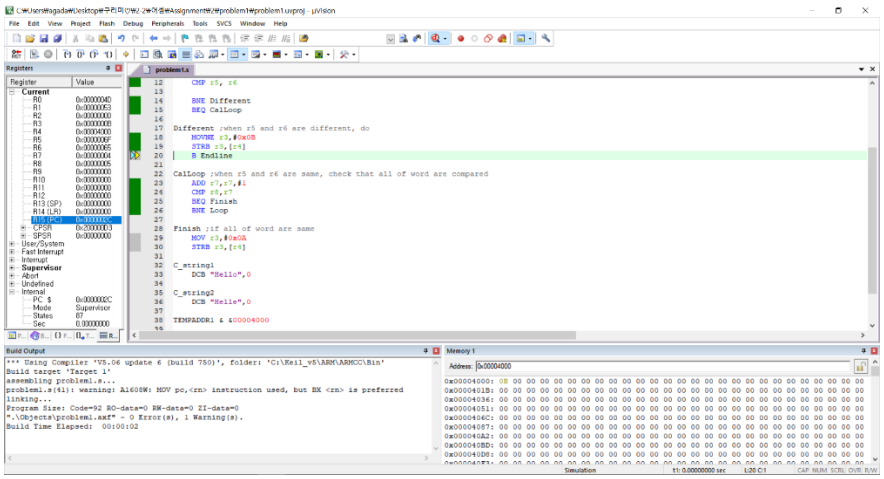
그 다음, Loop문으로 처음 들어가게 되면, 메모리 r0, r1번지에 저장된 값을 한 바이트 단위로 읽고 각각 r5, r6에 대입된다. 이는 한 단어 씩 각각의 레지스터에 저장된 것을 의미한다. 그리고 이 두 레지스터를 비교하여 다르면 Different 부분으로, 같으면 CalLoop부분으로 가게 지정한다.



"Hello"의 첫번째 알파벳인 'H'와 "Helle"의 첫번째 알파벳인 'H'가 같기 때문에 CalLoop문으로 들어왔다. 다섯 글자이므로 이 두 단어가 같은 지 알기 위해서는 네 번의 비교가 더 필요하다. 이에 r7의 값을 하나 올리고, r8과 비교해 같지 않음으로 다시 Loop문으로 돌아가게 지정한다.

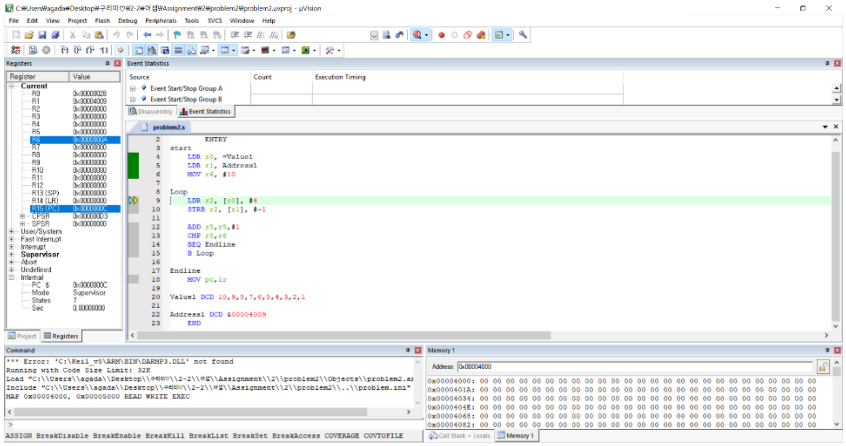


위의 과정을 3번 더 반복해 네번째 자리의 알파벳까지 비교한 후, 다섯 번째 알파벳을 비교하게 되면, 'o'와 'e'가 다르기 때문에 r5와 r6에 각각 다른 값이 저장이 된 것을 확인할 수 있다. 이에, Different 구문으로 가게 된다.

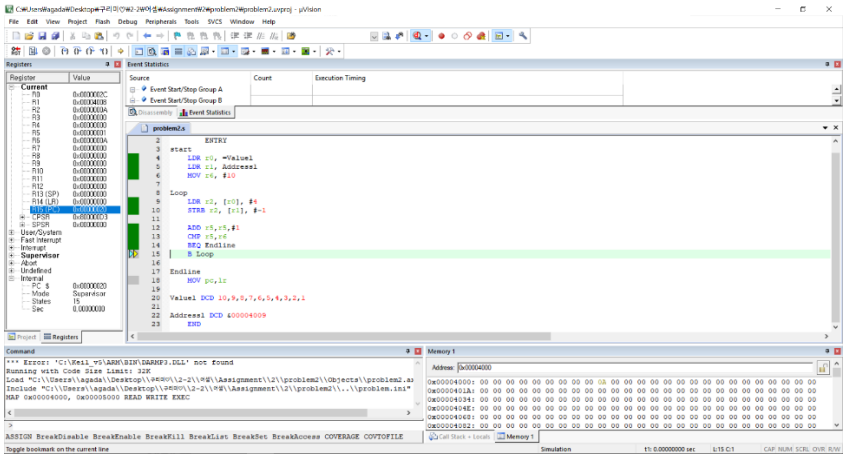


Different 구문으로 들어와서 r3에 0x0B가 대입되고, 이를 r4에 저장된 메모리의 주소에 r3에 들어있는 0x0B가 저장이 된다. 즉, 0x00004000번지의 메모리에 0B가 저장된 것을 확인 할 수 있다. 마치고 Endline으로 가게 된다.

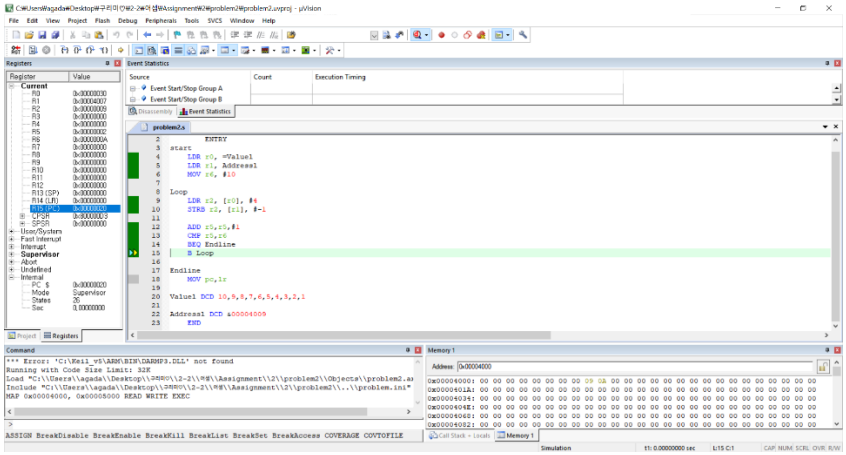
2. Problem2



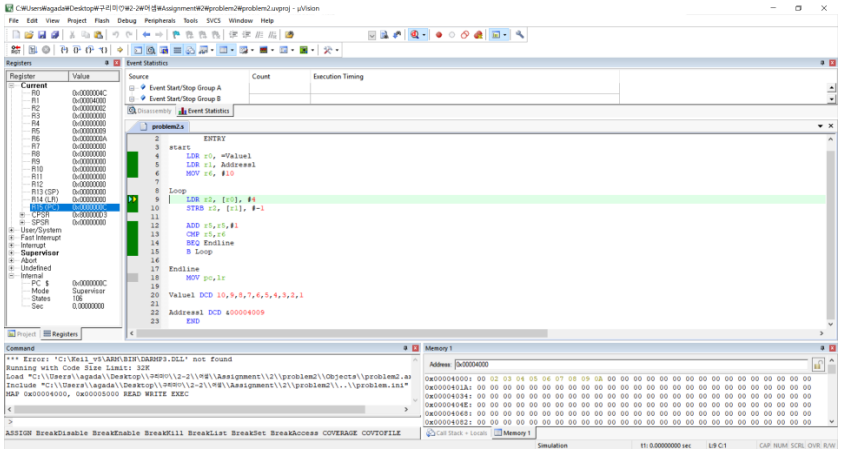
Start 구문에서 r0에 Valuel 배열의 첫번째 주소 값이 저장되고 r1에 Addressl, 아래에서 지정한 번지의 주소 값이 저장되었다. 끝으로 r6에는 비교해야하는 횟수인 10이 저장된 것을 왼쪽 registers화면에서 확인할 수 있다.



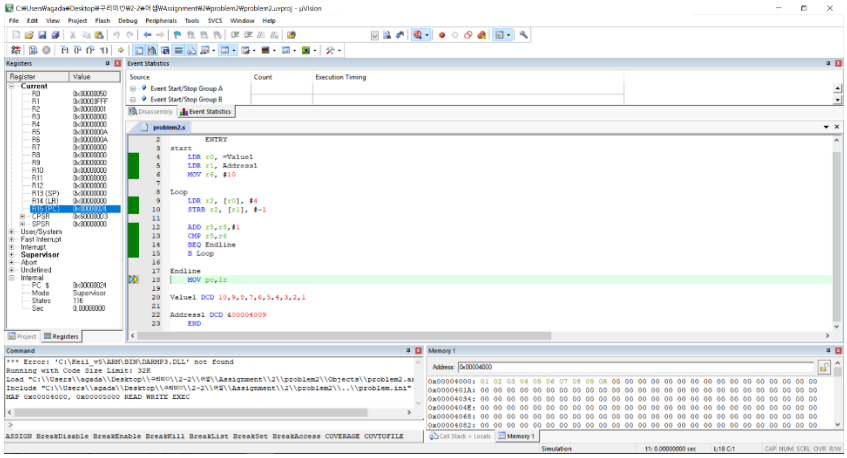
Loop문을 처음 돌게 되면 배열의 첫번째 숫자를 r2로 읽어와 0x0000000A가 저장된 것을 볼 수 있고 이를 r1에 저장된 메모리 번지에 저장해서, 0x00004009에 0A가 저장되었으며, r0는 4증가하고, r1은 1 감소한 것을 확인할 수 있다. 이후 r5를 1증가시켜 비교 숫자인 r6와 비교해 다르므로 Loop로 돌아갈 수 있게 해준다.



위와 마찬가지로, Loop를 한 번 더 돈 결과는 위와 같이 0x00004008에 9가 저장이 되고 r0, r1, r5는 4, 1, 1씩 증가한 것을 확인 할 수 있다.

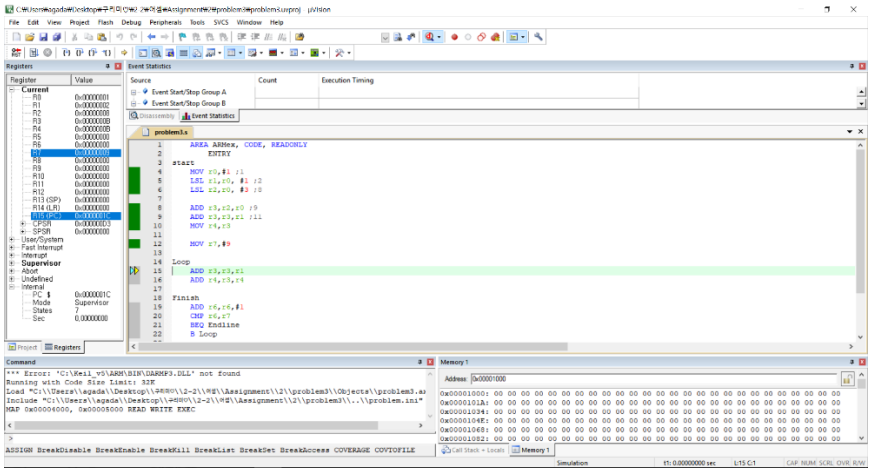


위의 과정을 7번 반복하게 되면 0x0004001~0x00004009번지의 메모리에 배열의 숫자들이 들어간다.

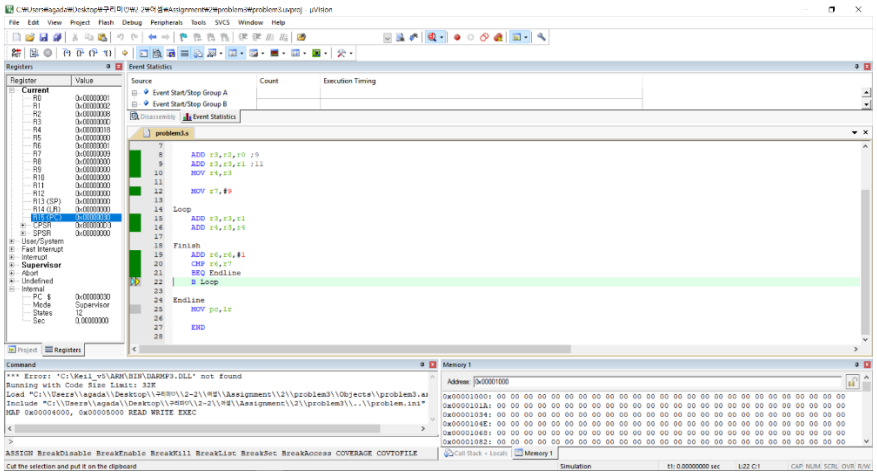


이때, Loop를 실행하게 되면 마지막으로 0x00004000에 01이 들어간 후 r5의 값이 하나 증가해, 10이된다. 이는 r6의 값과 같아져 Endline으로 이동하게 된다.

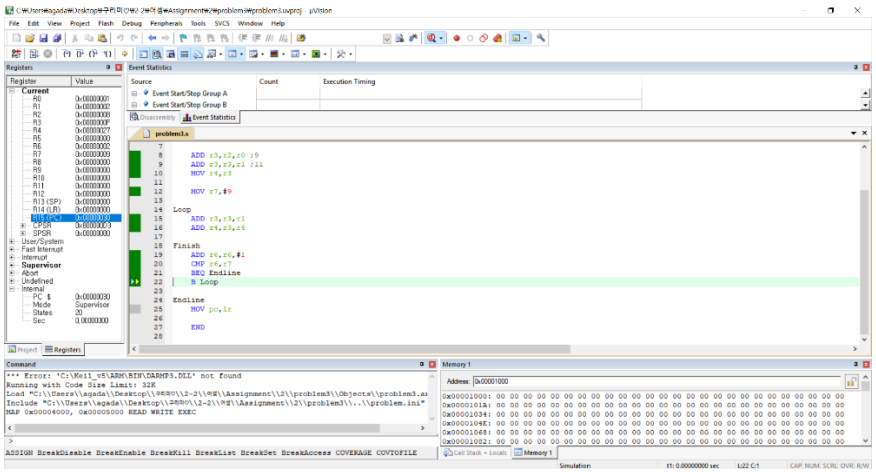
3. Problem3_1



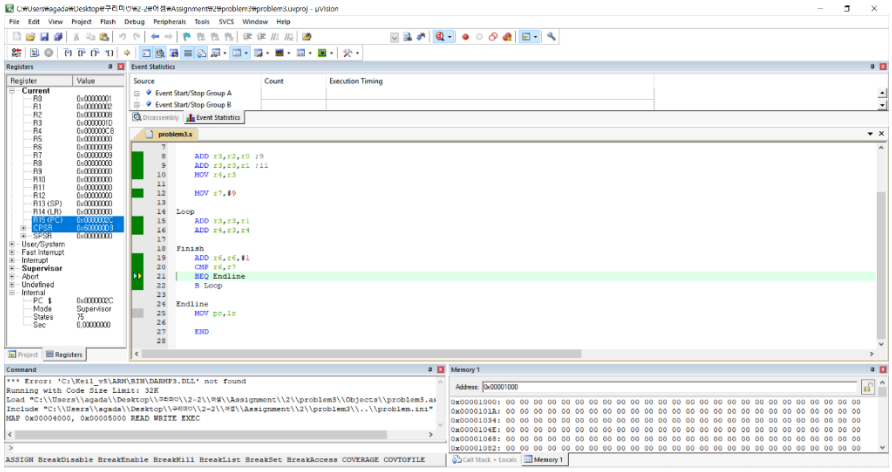
Debug를 시작하면 start 구문을 처음으로 만나게 된다. 이 구문에서는 r0에 1을 넣고 LSL연산을 통해 r1과 r2에 각각 2,8의 값을 대입하게 된다. 그리고 11이란 숫자를 만들기 위해서 r3에 r0,r1,r2를 더한 값을 대입하고, r4에 이 값을 복사 시킨다.



첫번째 Loop를 만나게 되면, r3에 2를 더한 값을 r3에 넣고, r4에 r3를 더해 저장한다. 이는 11과 13을 더해 r4에 저장한 것을 의미한다. 다음 Finish문으로 가서 r6를 하나 증가시키고, 비교 숫자인 r7와 비교하여 아직은 해야 할 연산이 남아있으므로 Loop문으로 돌아가게 된다.

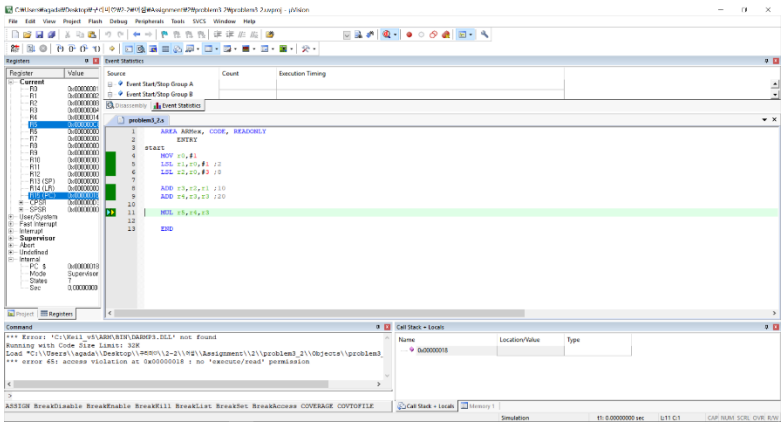


두번의 Loop와 Finish를 거치게 되면, r3는 15가 되고 r4에는 원래 저장된 11+13에 15를 더한 11+13+15의 값이 저장된 것을 확인할 수 있다. 이후, 할 연산이 더 있는지 확인하는 Finish문을 거치고 r6와 r7이 같아질 때까지 연산을 계속하게 된다.



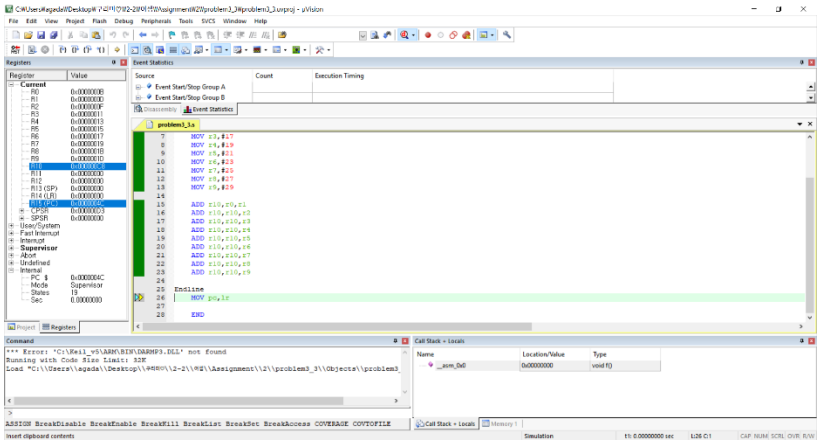
일곱번의 연산을 더 거쳐 r3가 29가 되면, r4에는 11+ 13+ 15+ 17+ 19+ 21+ 23+ 25+ 27+ 29가 저장된다. 고로 해야 할 연산이 끝났으므로 r6의 값이 r7과 같아져서 Endline으로 가게 된다.

4. Problem3_2



r0에 1이 저장되고, LSL연산을 통해 r1, r2에 차례로 2와 8이 저장된다. 이 두 값을 통해 r3에는 10, r4에는 20을 저장한다. 등차수열의 합의 공식에 따라서 r5에 r3*r5를 저장하면, 이가 곧 11+13+…+29의 결과인 0xC8이 나온 것을 확인 가능하다.

5. Problem3_3



r0~r9에 11,13~29를 대입, 이 모두를 r10에 더해준다.

3. 고찰 및 결론

A. 고찰

Problem1에서 c언어등 많은 high-level language에 존재하는 문자열 비교함수, strcmp를 구현하는 것이었다. 수업시간에 배열에 대해서 배웠고 문자열도 비슷한 원리로 구성되는 것은 알았지만 자세한 구현에 대해서는 배우지 않았기 때문에 문자열 자체를 선언하는 데부터 문제가 되었다. 전 과제에서 느낀 바에 따라서 인터넷과 강의자료를 살펴보았다. 그 결과, 문자열 생성 방법을 알게 되어 이를 이번 과제에 적용하였다.

다음으로는 익숙하지 않았던 label의 사용이 문제가 되었다. 여태까지 배웠던 다른 언어들은 모두 if문이나 while문 등 정해진 이름의 문들을 사용하였다. 하지만 어셈블리 언어에서 label의 이름은 본인이 정해야 한다는 것을 알지 못했었다. 검색 과정에서 전의 문제였던 string에 대해 알게 된 페이지에서 설명이 되어 있는 것을 알게 되었고 이를 읽고 label의 원리와 의미에 대해서 이해했다. 이를 응용해 프로그램을 완성했다.

B. 결론

아래 performance에서도 언급하지만, States 수로 problem3의 3가지 방식에 대한 performance를 판단하였을 때, 2번째 방법이 가장 효율적이고 그 다음은 unrolling 방법이었고 가장 효율성이 떨어지는 코드는 1번째, Loop문을 이용한 것이었다. 하지만, 숫자가 더 많아지면 많아질수록, 사용할 수 있는 레지스터의 개수는 한정되어 있으므로 3번은 사용이 어려워진다는 점, 같은 등차를 가지고 초항만 달라지면 코드를 고치는 것이 까다로워진다는 점에서 3번째 방법은 문제가 있다고 생각했다.

이번 과제에서 새로이 Conditional execution과 branch를 사용하는 구문에 대해서 다루게 되었다. 아래 두 코드를 보며 이 두 방식의 차이점을 설명하면, 다음과 같다.

■ Conditional execution

```
    AREA ARMex, CODE, READONLY
        ENTRY
Main
    LDR r0, =Value1
    LDR r1, Address1
    MOV r6, #6              ;입력 값 개수

Loop
    LDR r2, [r0], #4
    CMP r2, #0              ;x = 0?

    ;x>=0
    MULGE r3, r2, r2        ;r3 = x^2
    MULGE r4, r3, r3        ;r4 = x^4
    MULGE r7, r3, r2        ;r7 = x^3
    MULGE r8, r4, r2        ;r8 = x^5
    SUBGE r2, r8, r7        ;r2 = x^5 - x^3

    ;x<0
    MULLT r3, r2, r2        ;r3 = x^2
    MULLT r4, r3, r3        ;r4 = x^4
    MULLT r7, r3, r4        ;r7 = x^6
    ADDLT r2, r7, r4        ;r2 = x^6 + x^4

    STR r2, [r1], #4
    ADD r5, r5, #1          ;loop 수 계산
    CMP r5, r6              ;loop수 = 6?
    BEQ Endline             ;6번 loop -> 종료
    B Loop                  ;loop

Endline
    MOV pc, lr

Value1 DCD 3, 2, -3, -2, 10, 0
    ;DEC   216, 24, 810, 80, 99000, 0
    ;HEX    D8, 18, 32A, 50, 182B8, 0
Address1 DCD &40000000
    END
```

■ Using branch instruction

```
    AREA ARMex, CODE, READONLY
        ENTRY
Main
    LDR r0, =Value1
    LDR r1, Address1
    MOV r6, #6              ;입력 값 개수

Loop
    LDR r2, [r0], #4
    CMP r2, #0              ;x = 0?

    BLT Less
    ;x>=0
    MUL r3, r2, r2  ;r3 = x^2
    MUL r4, r3, r3  ;r4 = x^4
    MUL r7, r3, r2  ;r7 = x^3
    MUL r8, r4, r2  ;r8 = x^5
    SUB r2, r8, r7  ;r2 = x^5 - x^3
    B Finish

Less ;x<0
    MUL r3, r2, r2  ;r3 = x^2
    MUL r4, r3, r3  ;r4 = x^4
    MUL r7, r3, r4  ;r7 = x^6
    ADD r2, r7, r4  ;r2 = x^6 + x^4

Finish
    STR r2, [r1], #4
    ADD r5, r5, #1          ;loop 수 계산
    CMP r5, r6              ;loop수 = 6?
    BEQ Endline             ;6번 loop -> 종료
    B Loop                  ;loop

Endline
    MOV pc, lr

Value1 DCD 3, 2, -3, -2, 10, 0
    ;DEC   216, 24, 810, 80, 99000, 0
    ;HEX    D8, 18, 32A, 50, 182B8, 0
Address1 DCD &40000000
    END
```

두 코드는 같은 의미를 담고 있다. 하지만 이 두 코드를 직접 실행해 보면 states

가 후자의 것이 적게 나오는 것을 확인할 수 있다. 그리고 이에 따라 성능에서 using branch execution이 더 좋다는 것을 알 수 있다.

이유는, conditional execution을 사용하면, 비교함수를 이용해 크다는 것을 확인한 후에도 작을 때 실행하는 코드 또한 거쳐야 한다는 단점이 있기 때문이다.

4. Performance

이번 과제에서는 같은 문제를 세가지의 방식으로 프로그래밍하였다. 그리고 그 결과 어느 방식의 performance가 더 효과적인지 알아보기 위해서 states의 수를 비교하여 본다.

A.Problem 3_1

States: 76

B.Problem 3_2

States:8

C.Problem 3_3

States:19

States 수로 판단하였을 때, performance는 2번째 방법이 가장 효율적이었다. 등차수열 공식을 이용하여 코딩한 방법이었다. 그 다음은 unrolling방법이었고 가장 효율성이 떨어지는 코드는 1번째, Loop문을 이용한 것이었다.

5. 참고문헌

String & Label /
http://lanian.tistory.com/entry/%EC%A0%84%EA%B4%91%EC%84%B1%EC%9D%98-%EC%96%B4%EC%85%88%EB%B8%94%EB%A6%AC%EC%96%B4-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B03%ED%9A%8C-%EC%96%B4%EC%85%88%EB%B8%94%EB%A6%AC%EC%96%B8%EC%96%B4-%EA%B8%B0%EC%B4%88-2

이형근/ 어셈블리프로그래밍/ 광운대학교/ 2018