어셈블리 프로그래밍 설계 및 실습
보고서

실험제목: Block Data Transfer & Stack

실험일자: 2018년 10월 09일 (화)

제출일자: 2018년 10월 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

Block Data Transfer & Stack

B. 목적

Block Data Transfer : 레지스터와 메모리간 블록 단위의 데이터 저장/ 가져오기를 이해한다. 또한 어셈블리어의 서브 루틴(함수) 사용 방법에 대해 익힌다.

Stack : 프로그래밍 수준에서 스택 명령어의 필요성과 효용성에 대해 이해하고 응용한다. 여러 종류의 스택 저장 방식(stack type)에 대해 이해하고 각 방식에 다른 메모리 접근 방법을 실질적으로 확인해봄으로써 그 차이를 이해한다.

2. 설계 (Design)

A. Pseudo code

1. Problem1

[sp]=R0
[sp+1]=R1
[sp+2]=R2
[sp+3]=R3
[sp+4]=R4
[sp+5]=R5
[sp+6]=R6
[sp+7]=R7

R1=[sp]=기존R0
R6=[sp+1]=기존R1
R0=[sp+2]=기존R2
R2=[sp+3]=기존R3
R7=[sp+4]=기존R4
R3=[sp+5]=기존R5
R4=[sp+6]=기존R6
R5=[sp+7]=기존R7

2. Problem2

- start

  R0~R7=0~7

  [sp~sp+7]=R0~R7

  R9=160

- doRegister()

  R0=R0+0

  R1=R1+1

  R2=R2+2

  R3=R3+3

  R4=R4+4

  R5=R5+5

  R6=R6+6

  R7=R7+7

  R8=R0+R1+R2+R3+R4+R5+R6+R7

- doGCD()

```
while( r8 != r9 ) do
{
        If(r8 > r9)
                r8 = r8 – r9
        else
                r9 = r9 - r8
}
```

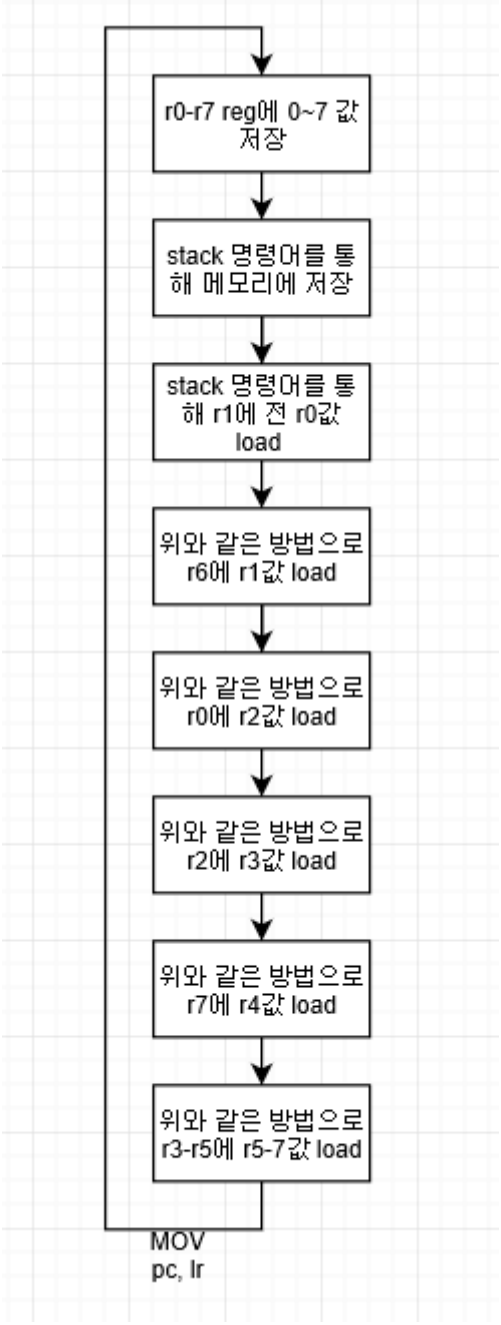
- saveToStack()

```
for(i=0 ; i<8 ; i++)
{
        r10 = [sp + 7 – i]
        r[i] = r[i] +r10
}
```

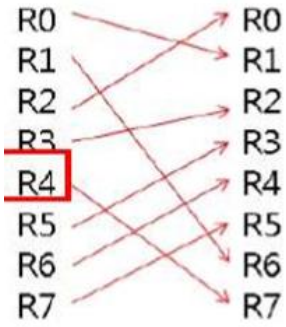
```
[sp]=r8 (gcd결과값)
for(i=1 ; i<8 ;i++)
{
        [sp + i] = r[i]
}
```

B. Flow chart 작성

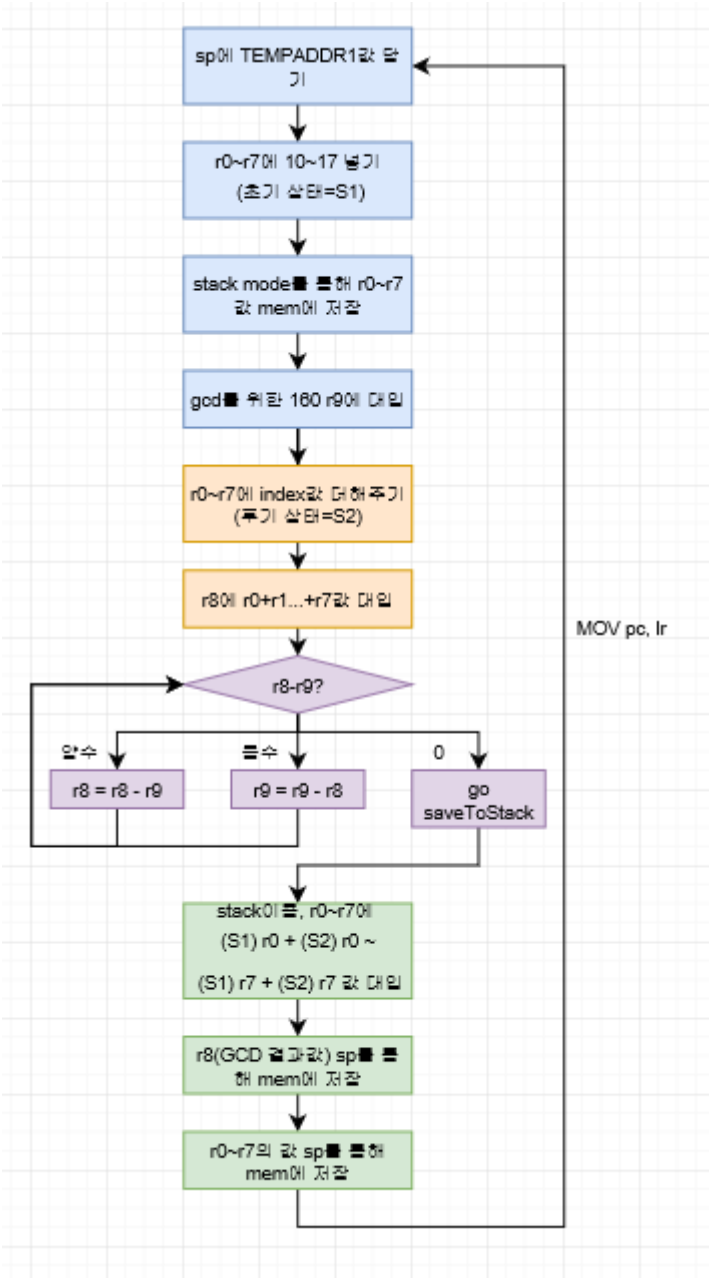
1. Problem1



R0~R7의 레지스터에 0~7의 값을 대입해 준다. 후에 이 register값들을 STMFD 즉, FULL Descending으로 sp를 통해 R0~R7을 차례대로 저장한다. 후에 LDMFD을 통해서 저장했던 값들을 아래 그림과 같이 알맞은 순서대로 저장한다.



2. Problem2



- start

  초기값들을 넣어주는 함수이다. sp에 저장 주소인 0x40000을 넣어주고 r0에 10을, r1에 11을 넣고 하나씩 증가시켜 register에 값을 넣어주면 r7=17까지 대입이 된다.

- doRegister

  ADD 명령어를 통해서 초기 register의 값들의 집합인 S1에 각각 레지스터의 index의 숫자들을 더해준다. (예를 들어, r0+0, r1+1, r2+2 ,,,.) 그리고 결과적으로 나온 후기 register의 값들을 S2라 명명한다. S2의 요소들을 모두 더한 값을 r8에 대입해준다.

- doGCD

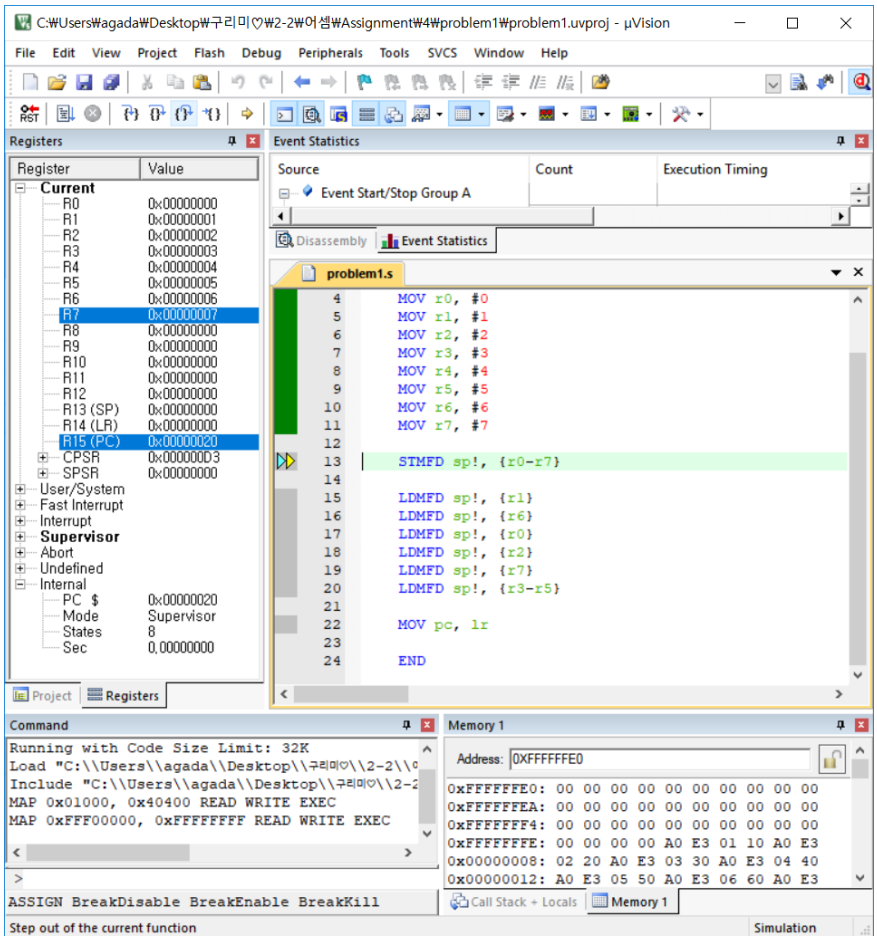
  r8과 r9의 최대공약수를 구해주는 함수이다. r8과 r9의 값을 비교해 r8이 작으면 r9에 r9에서 r8을 뺀 값을 대입시켜주고 크면, r8에 r8에서 r9을 뺀 값을 대입해준다. 이를 계속하다 같아질 시에 saveToStack으로 넘어간다.

- saveToStack

  구한 값들을 바탕으로 stack으로 memory에 채워주는 함수이다. S0의 r0과 S1의 r0의 합을 r0에 대입, r1~r7도 같은 방법으로 진행한다. 이후 r8 즉, GCD의 결과값을 sp(0x40000)에 넣어준다. 다음으로 r0~r7의 값을 sp(0x40004~40020)에 넣어준다.

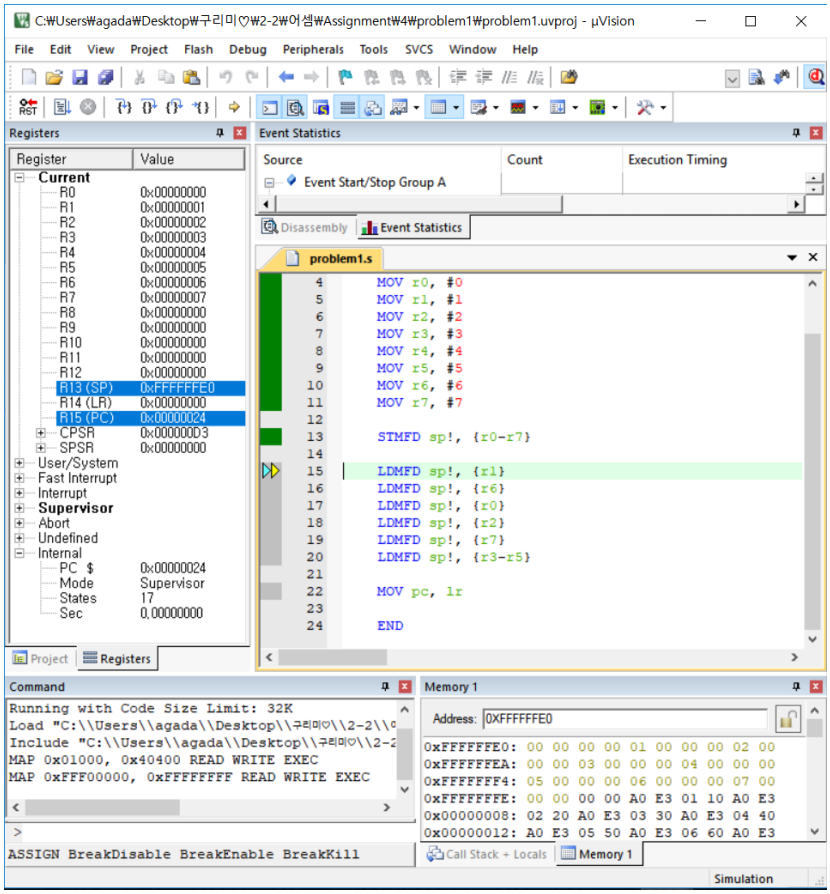
C. Result

1. Problem1

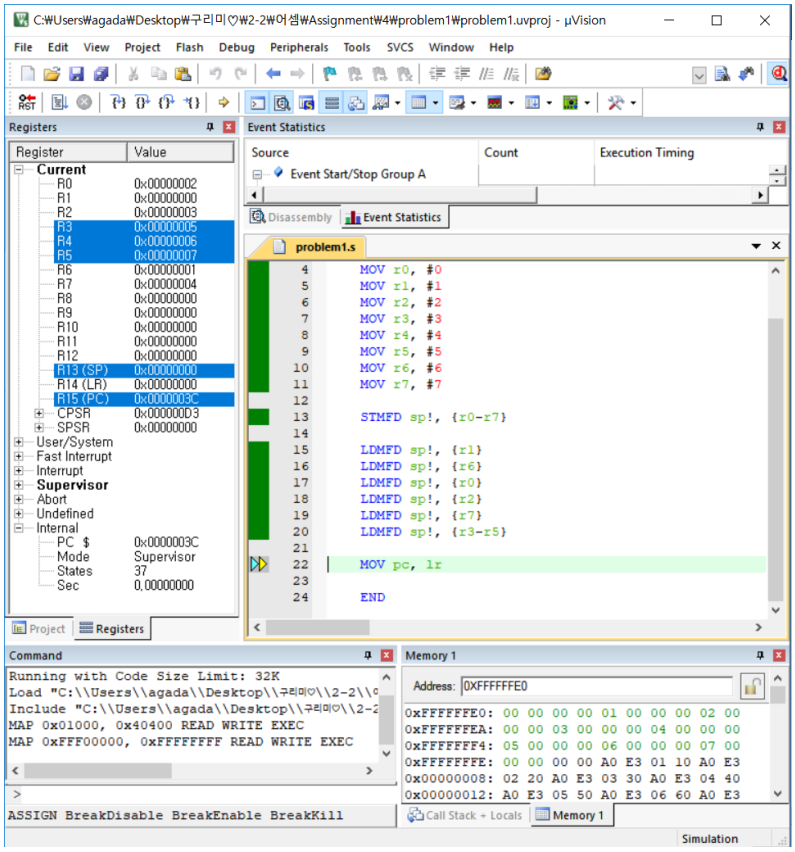


Register의 값들을 초기화 시켜주는 작업이다. 별도로 공지된 저장 값이 없으므로 본인은 r0~r7에 순서대로 0~7의 값을 대입했고 결과적으로 위의

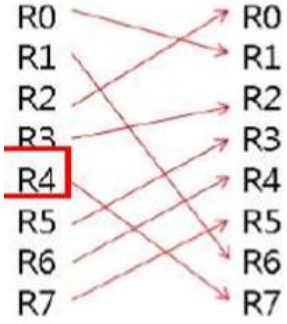
결과창이 뜨게 된다.



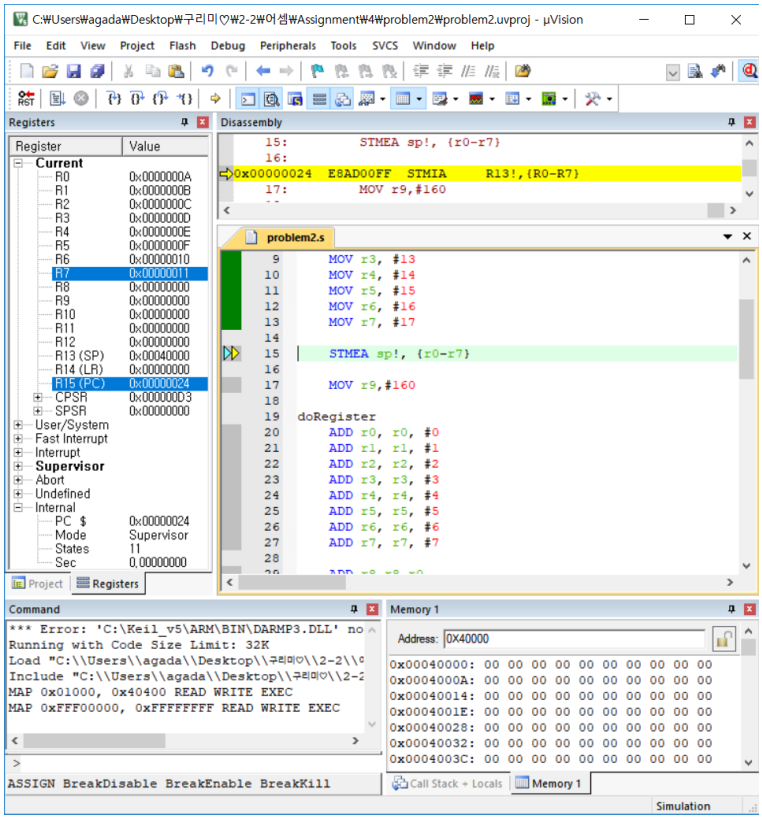
별도의 sp를 지정하지 않고 full descending방법으로 r0~r7의 값을 저장하였다.



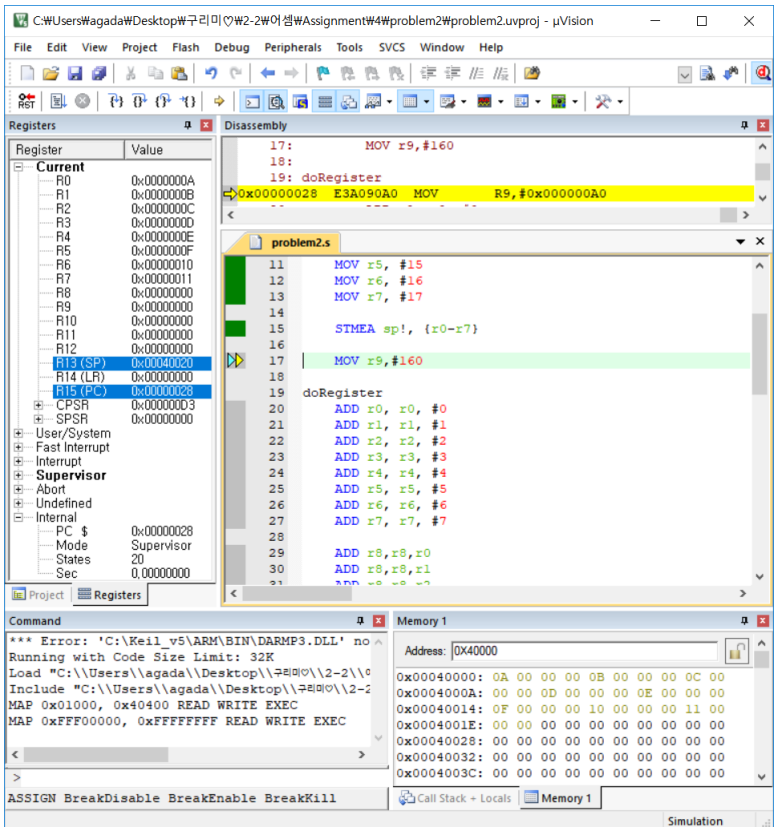
후에 아래 그림과 같은 순서대로 mem에 있는 값을 register로 옮기는 과정을 수행하였다. 그 결과 위의 그림과 같은 화면이 떠오른다.



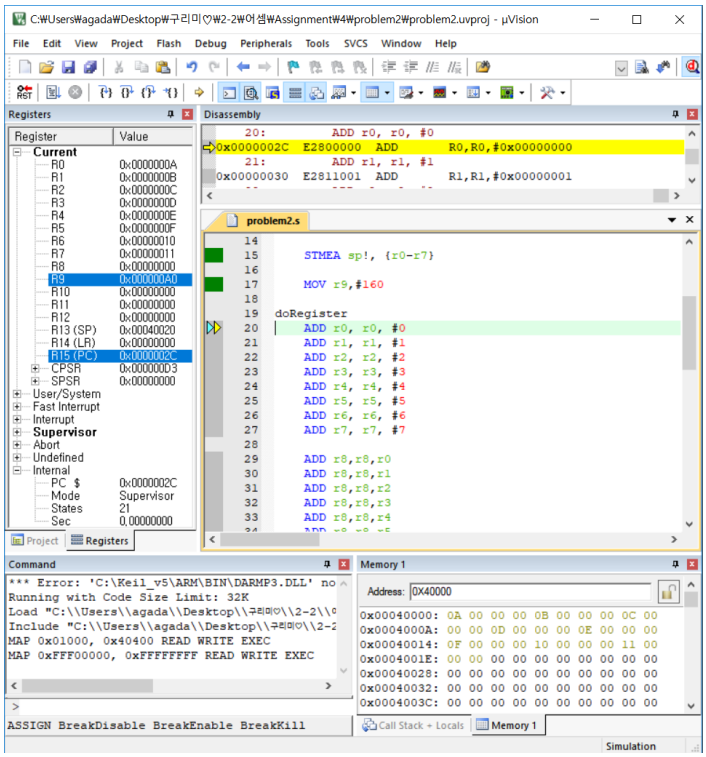
2. Problem2



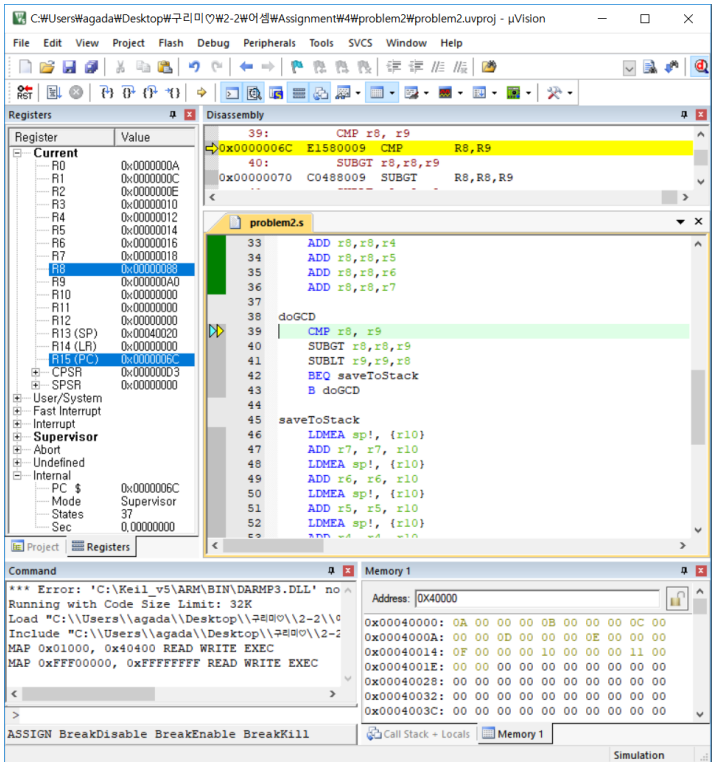
R0~R7의 값을 10~17로 초기화하였다.



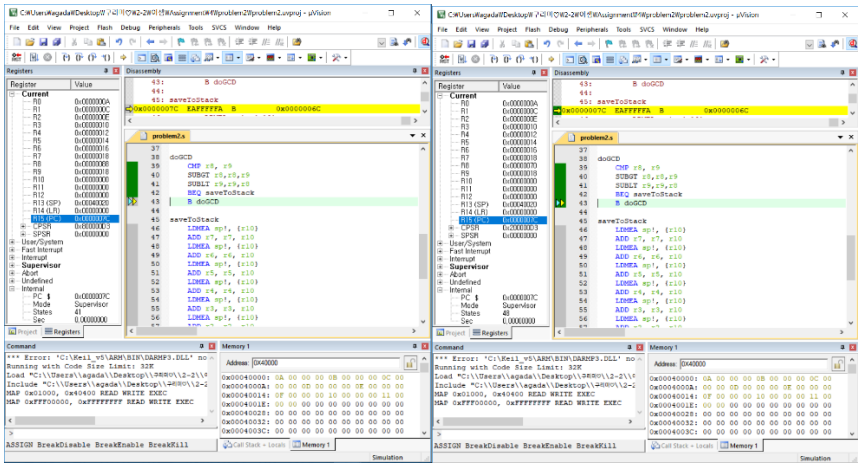
위에서 sp에 TEMPADDR1의 값을 저장했다. 그 값을 바탕으로 STMEA의 방식으로 R0~R7의 값을 저장시켰다. (R0~R7의 값을 바꿔줄 것이기 때문) 결과적으로 위의 화면이 나타난다.

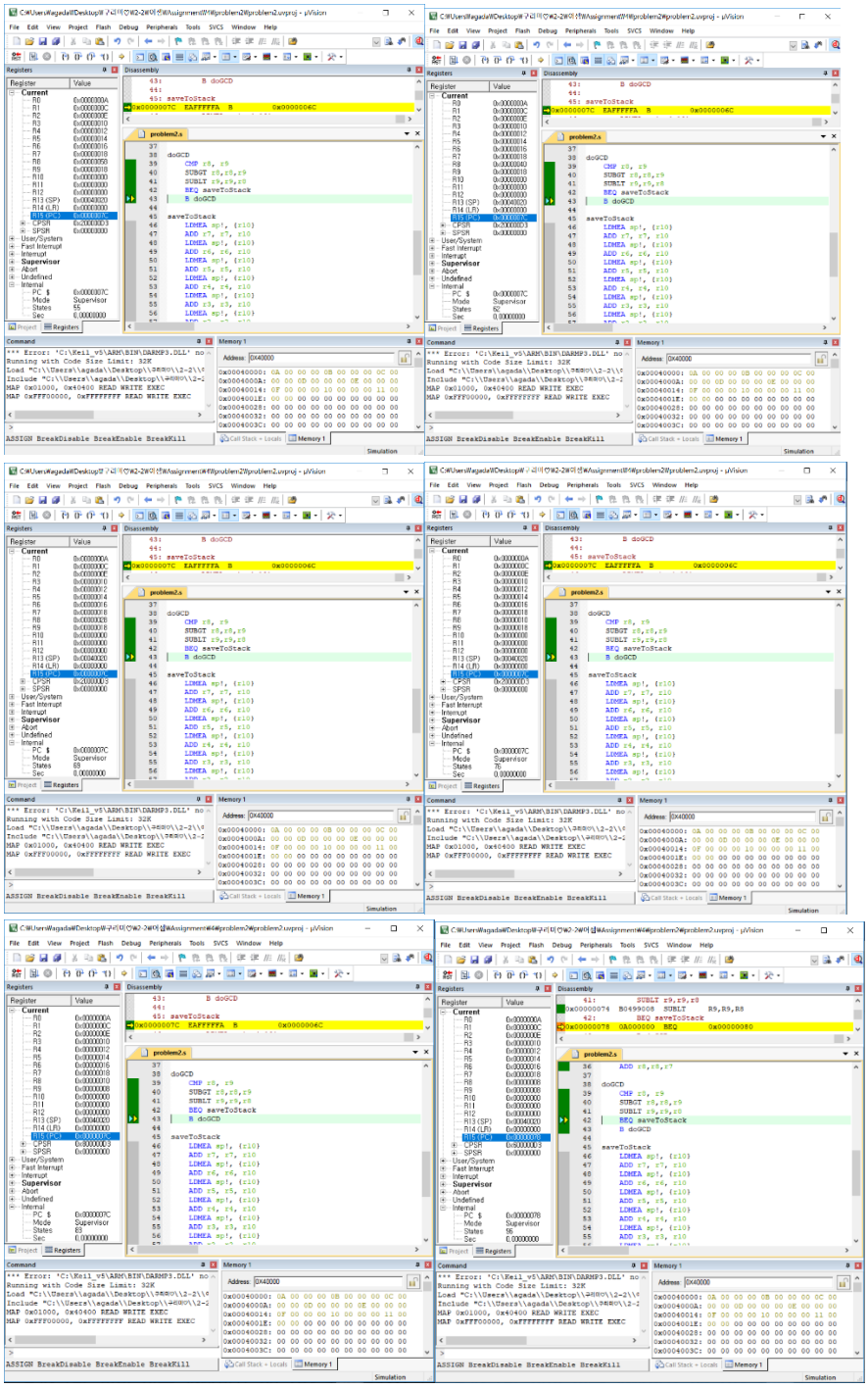


R9에 gcd비교값인 160을 대입하였다.

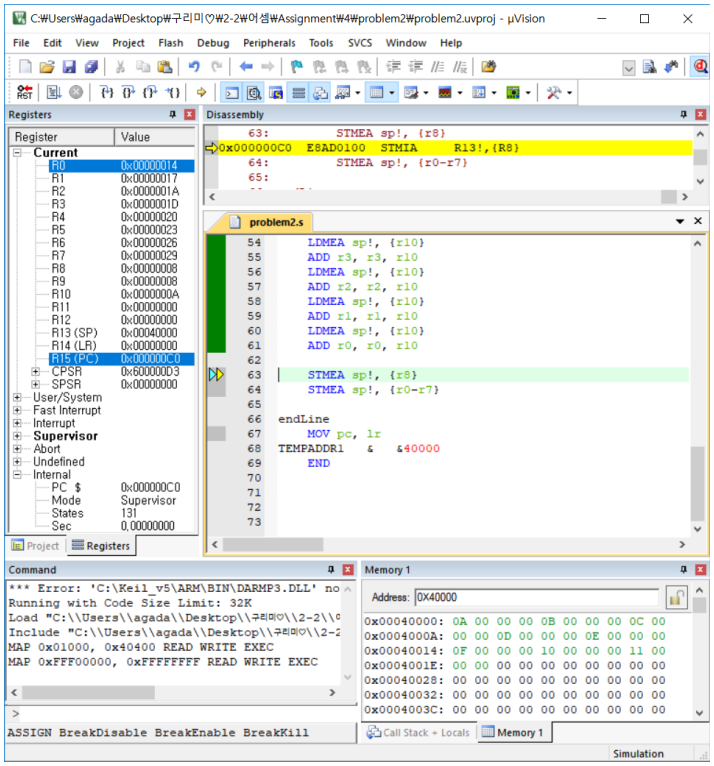


doRegister에 들어가 R0~R7의 값에 0~7을 더하고 R8에 R0~R7의 합을 저장하였다.

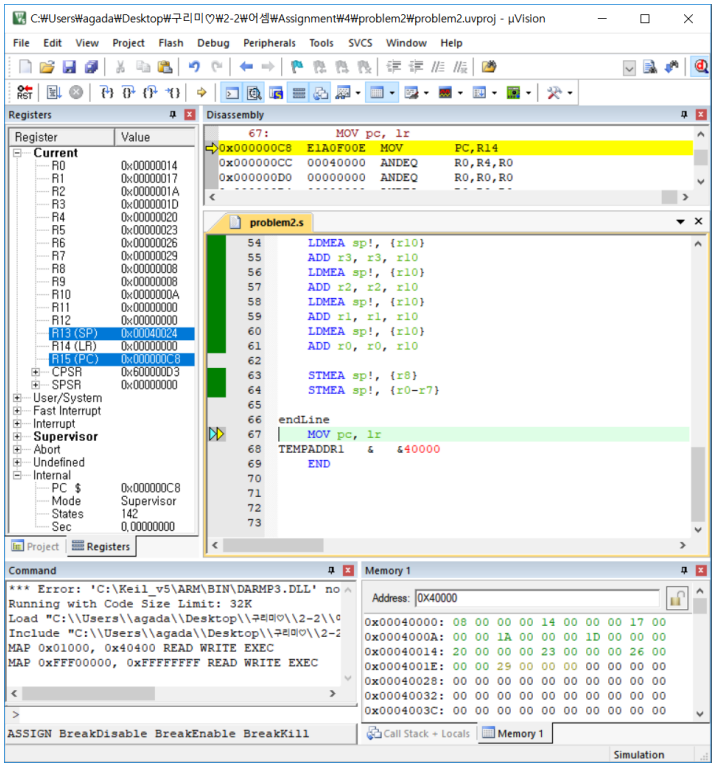




GCD를 구하기 위해 branch instruction을 통해 LOOP를 돌아 결과값을 만들어 냈다.



STMEA로 저장했던 기존 값과 register에 저장되어 있는 새로운 값을 더한 값을 register에 저장한다.



STMEA를 통해 차례로 GCD값, R0~R7의 값을 sp로 저장시킨다.

3. 고찰 및 결론

A. 고찰

1) Stack을 이용한 sorting

Problem1은 이번시간에 배운 stack pointer활용법을 통해 register에 있는 값들을 memory에 옮긴 후 원하는 register의 위치로 각 값들을 옮기는 과제였다.

강의시간에 강의를 듣고 실습을 진행하며 이해를 대충했지만 LDMFA와 STMFA의 FULL/EMPTY의 상황과 ASCENDING/DESCENDING이 반대인 것을 확실히 이해하지 못한 상태였기 때문에 LDMFA와 STMED를 사용하여 값을 store, load하려했다. 하지만 생각대로 진행되지 않는 것을 확인했고 강의 자료를 보며 다시 공부를 했다. 이에 LDMDA=LDMFA이고 LDM과 STM이 쌍을 이루기 위해 F/E, A/D가 반대라는 사실을 깨닫게 되었고 코드를 수정하였다. 곧, 올바른 결과값을 얻어낼 수 있었다.

2) do Register & do GCD

Problem2는 branch명령어를 통해서 register의 값들을 바꾸고, register의 값을 이용해 160이라는 숫자와의 최대공약수를 구한 후, 이들을 memory에 저장하는 과제였다.

이 과제에서는 다른 요소들은 수월히 진행되었으나 gcd를 구하는 방법면에서 문제가 있었다. 평소에 gcd를 구할 때는, 나눗셈을 이용하여 두 수의 공통인 소인수로 각 수를 나누고 몫은 아래 식으로 내리고 후에 몫이 서로소가 될 때까지 계속 나누고, 공통으로 나온 소인수를 모두 곱하는 방법을 사용했다. 하지만 이렇게 구현을 하면 너무 복잡해질 것이라는 판단 하에 다른 방법으로 gcd를 구하는 방법을 검색했다. 결과적으로 위의 방법을 찾아내게 되어 performance적으로 더 좋은 code를 구현할 수 있었다.

B. 결론

1) Stack을 통한 sorting

Stack mode에 대한 정리표

Mode	POP	=LDM	PUSH	=STM
Full ascending (FA)	LDMFA	LDMDA	STMFA	STMIB
Full descending (FD)	LDMFD	LDMIA	STMFD	STMDB
Empty ascending (EA)	LDMEA	LDMDB	STMEA	STMIA

Empty descending (ED)	LDMED	LDMIB	STMED	STMDA

위와 같이 pop과 push모드에서는 LDM과 STM은 쌍을 이루고 STM이 Full/ Empty, Ascending/ Descending을 결정하고 LDM은 이 반대의 뜻을 지닌다고 생각하면 된다.

2) do Register & do GCD

```
int gcd(int a, int b)
{
  while (a != b) do
  {
   if (a > b)
         a = a - b;
   else
         b = b - a;
  }
  return a;
}
```

컴퓨터 프로그래밍으로 GCD를 쉽게 구현하는 방법은 위와 같다. GCD를 구한 두 수를 비교하여 같아질 때까지 큰 수에 큰 수 – 작은 수를 넣어주고 최종적으로 같을 때, 그 수가 최대공약수가 된다.

4. Performance

5. 참고문헌

이형근/ 어셈블리프로그래밍/ 광운대학교/ 2018