Report of Computer Architecture (MIPS Assembly)

임유택 (201514768)
전북대학교 컴퓨터공학부
j75575863@naver.com

Problem 1

1. 실습 프로그램의 구성 및 동작 원리

이 문제는 처음에 stack을 이용해 구현했고 간단한 반복을 이용해 더 간단히 할 수 있을 것이라 판단하여 반복문으로 간결하게도 구현해보았다.

(1) stack을 이용해 구현한 소스 :
stack을 이용한 구현에서는 stack에 store된 $t0를 load해 $v0를 더한 후 $v0를 1씩 줄이고 그 value를 다시 $t1에 넣어 stack에 store하도록 구현했다. 그 이후에는 할당 받은 stack을 return해주었다. 처음에 코드를 짤 때 루프를 빠져나가면 Higher-Level Language Program들처럼 변수에 저장되어 있는 것이 아니기 때문에 result value가 저장되지 않아서 어려움을 겪었지만 이 부분을 stack을 이용해서 잘 해결했다.

(2) 간단한 반복문으로 구현한 소스 :
반복문으로 구현한 소스에서는 n이 들어간 $a0를 1씩 줄이며 줄일 때마다 &t0에 더해 결국 &t0안에는 1부터 n까지의 합이 들어가게 된다.

소스를 통한 구현 설명

```
14          addi $sp, $sp, -4
```

$sp에 4를 빼서 stack 방을 1개 만든다.

```
16
17          loop:
18
19          lw $t0, 0($sp)
20
21          add $t1, $v0, $t0
22          addiu $v0, $v0, -1
23          sw $t1, 0($sp)
24
25          bne $v0, $0, loop
```

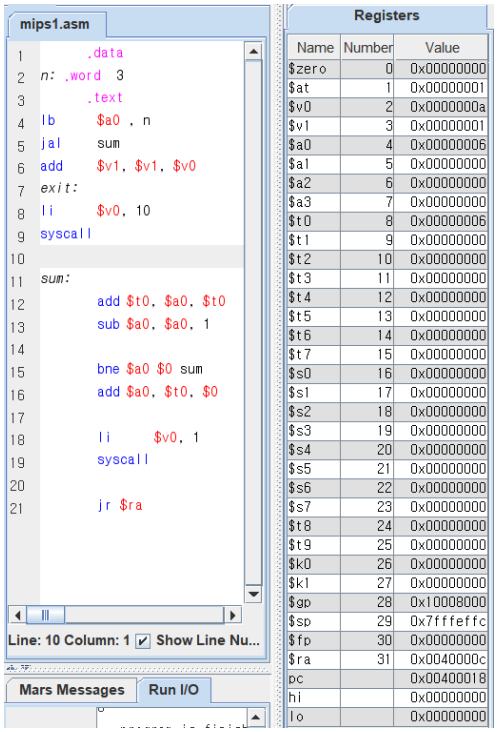
stack안에 store되어 있는 $t0를 load해 $v0를 더한 후 $v0를 1씩 줄이고 그 value을 다시 $t1에 넣어 stack에 store한다.

```
29          addi $sp, $sp, 4
```

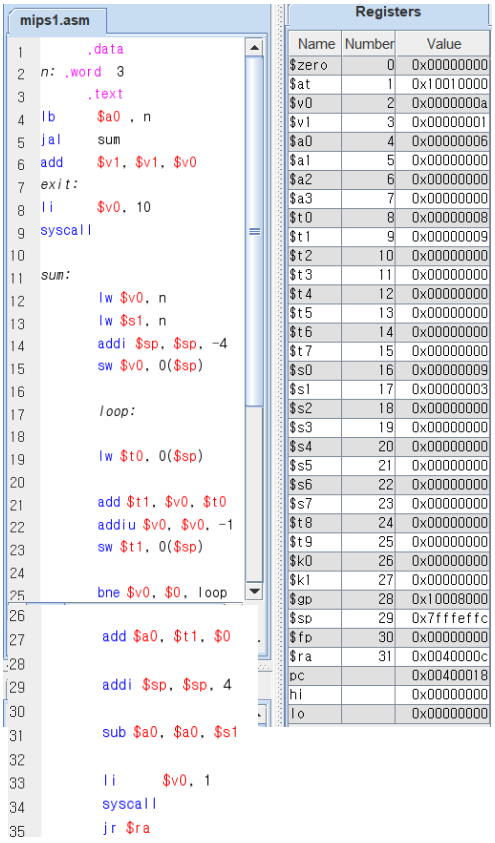
$sp에 4를 더해 만들었던 stack 방을 return한다.

2. 결과

간단한 반복문으로 구현한 소스



stack을 이용한 소스코드



둘 다 $a0에 n(3)까지의 합이 들어가 있는 것을 볼 수 있다.

3. 결론

1번 문제는 n까지의 합을 구하는 문제였고 stack을 이용해 구현하는 방법과 간단한 반복문을 이용해 구현하는 방법을 사용했다. stack을 이용한 소스코드는 MIPS 어셈블리어 코딩이 처음이라서 $sp를 이용하는 것이 익숙하지 않아 발상에서 어려운 부분이 많았다. 다시 돌아가면 value들이 초기화 되는 오류도 있었고 그 부분에서 stack를 써보자는 생각이 들었다. stack을 이용해 load와 store을 해서 안전하게 value를 operate할 수 있었다. 간단한 반복문을 이용해 구현하는 방법은 어렵지 않게 바로 구현이 가능했다.

Problem 2

1. 실습 프로그램의 구성 및 동작 원리

이 문제는 코드가 돌아가는 동안 변하는 value들을 나열해 보면 3가지가 필요하다.

- 첫 번째는 입력 받은 n에서 1씩 줄어들며 0이 될 때 멈추는 value
- 두 번째는 지금까지의 합을 구할 때 사용했던 마지막 value, 다시 말하자면 세 번째가 지금까지의 합의 value라면 두 번째는 직전의 항까지의 합의 value
- 세 번째는 지금까지의 합의 value

ex) n = 4 일 때 value들

변수1	4	3	2	1	0
변수2	1	1	2	3	5
변수3	1	2	3	5	8

이 때 result value = 8 이 나오는 이유는 첫 번째 항과 두 번째 항을 생략하기 때문이다.
그래서 입력 받은 n의 value에서 2를 빼고 시작했다.

변수1	2	1	0
변수2	1	1	2
변수3	1	2	3

그러므로 n = 4일 때 result value = 3

소스를 통한 구현 설명

```
16  fib:
17      lw $s1, n
18      addi $t0, $s1, -1
19      beq $t0, $0, exit
20      addi $t1, $s1, -2
21      beq $t1, $0, twoterm
```

처음 fib로 가게 되면 $s1에 n의 value를 할당하고 n이 1과 2인지 확인한다. 왜냐하면 $s1에서 2를 빼고 시작할 것이기 때문에 1과 2의 filter과정을 생략하면 오류가 나게 된다.

(1) 1인 경우 : $a0에 입력을 1로 받았으므로 그대로 exit로 가서 종료하면 된다.
(2) 2인 경우 : twoterm으로 이동해서 $a0에서 1을 빼 종료한다.

```
12  twoterm:
13      addi $a0, $a0, -1
14      jr $ra
```

twoterm은 $a0가 지금 2인 상태에서 온 것이므로 1을 빼 fib문을 끝내게 구현한다.

```
22      addi $s1, $s1, -2
23      addi $v1 ,$v1, 1
24      addi $v0 ,$v0, 1
25
26      addi $sp $sp -16
27      sw $ra, 12($sp)
28      sw $s1, 8($sp) # n의 수
29      sw $v1, 4($sp) # 1번 째
30      sw $v0, 0($sp) # 2번 째
```

앞서 말한대로 $s1을 2를 빼고 시작을 한다. 그리고 처음 $v0와 $v1을 1로 초기화 시킨다.

$s0	n - 2
$v1	1
$v0	1

표로 보면 이런 상태이다.
그 후 Loop문을 돌 것이기 때문에 stack 방을 만든다. 변하는 변수들을 store할 방 3개에 $ra를 store할 방 1개를 추가로 4개의 방을 할당한다. 그리고 순서대로 stack 방에 넣어준다.

```
32      loop:
33
34      lw $s1, 8($sp)
35      lw $v1, 4($sp)
36      lw $v0, 0($sp)
37      add $t0, $v0, $0
38      addi $s1, $s1, -1
39      add $v0, $v1, $v0
40      sw $s1, 8($sp)
41      sw $t0, 4($sp)
42      sw $v0, 0($sp)
43
44      bne $s1, $0, loop
```

Loop문 안의 내용은 다음과 같다.

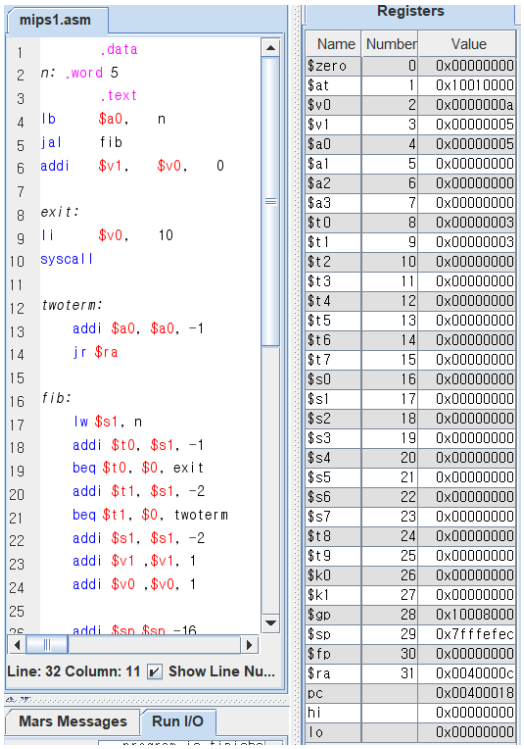
(1) stack 방에 들어가 있는 변수들을 load해 각각 $s1, $v1, $v0에 넣어준다
(2) $t0에 $v0의 value를 넣어준다. 왜냐하면 기존의 $v0의 value가 필요한데 $v0는 $v1과 $v0를 더한 value를 store할 것이기 때문이다.
(3) $s1은 1씩 줄어들고, (2)에서 말한대로 $v0에는 전 항까지의 합($v0)과 지금 항($v1)의 합을 구해서 넣어준다.
(4) stack 방에 각각 $s1, $t0, $v0를 store해준다
(5) 만약 1씩 줄어들던 $s1이 0이 된다면 이 Loop를 나간다.

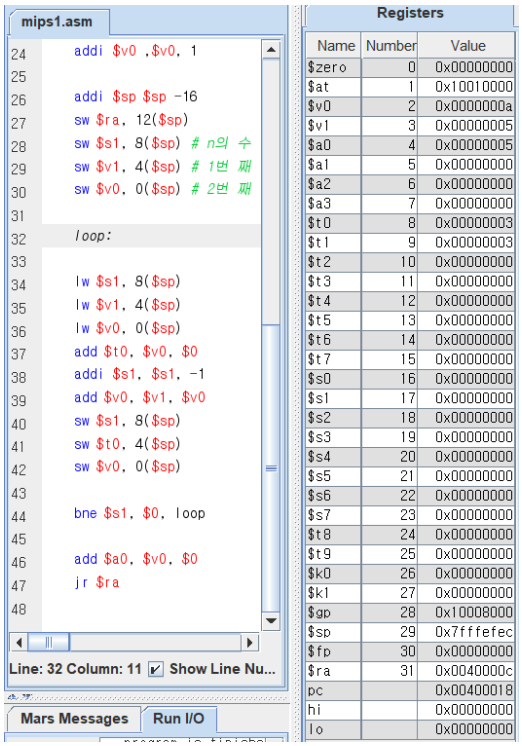
```
46      add $a0, $v0, $0
47      jr $ra
```

$v0에 있는 result value를 $a0에 넣어준다.

2. 결과

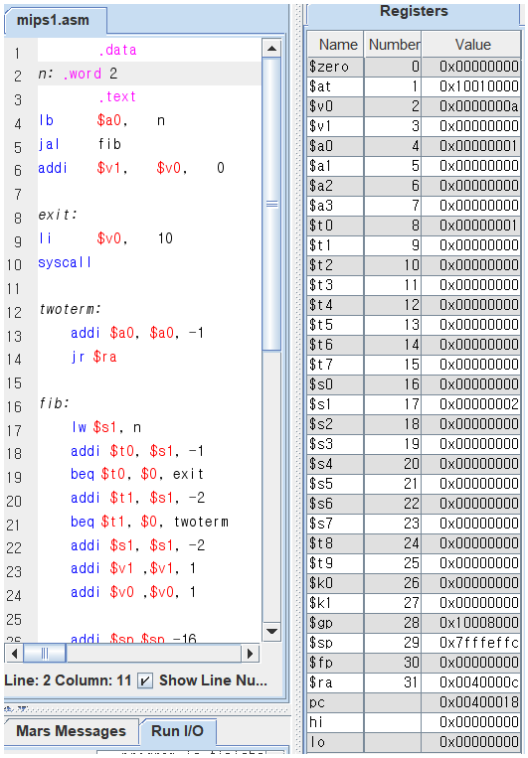
문제에서 주어진 대로 n = 5 일 경우





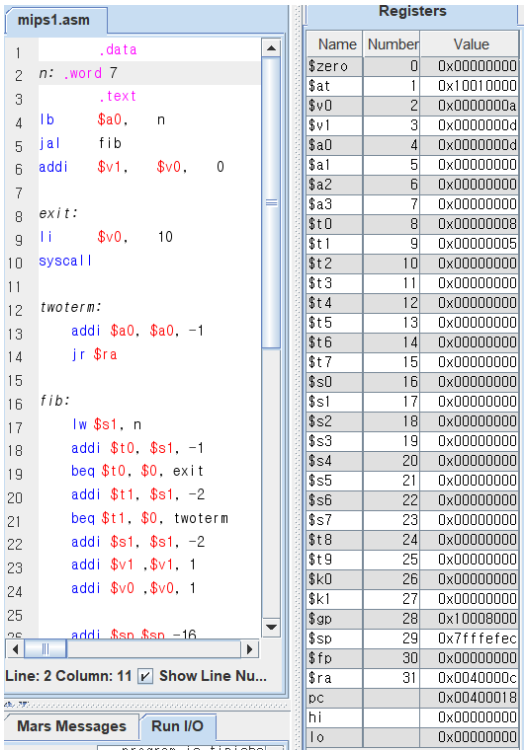
$a0가 5인 것을 확인할 수 있다.

n = 2 일 경우



$a0가 1인 것을 확인할 수 있다.

n = 7 일 경우



$a0가 d (16진법으로 13)인 것을 확인할 수 있다.

3. 결론

1번 문제에서 stack을 이용해 문제를 풀어보니까 별로 어려웠던 문제는 아니였다. 2번 문제는 피보나치 수열의 항을 구하는 문제였다. 알고리즘을 생각하고 계속 변하는 value들을 stack에 넣어 operate했고 문제를 해결했다. 구현했을 때 n의 value가 1과 2인 경우만 오류가 났고 3부터는 정상적인 피보나치 수열의 항이 나왔다. 그래서 생각해본 결과 1이 들어있는 첫 번째 항과 두 번째 항을 고려하지 않고 operate했었다. 그래서 n의 value가 1과 2인 경우를 exit처리를 해주었고 n이 1과 2인 경우에도 경우에 맞는 피보나치 수열의 항이 나왔다.