과제 #1 – MIPS Procedure Call 구현

김수환 (201510743)
전북대학교 컴퓨터공학부
suwhan77@naver.com

요약
엄준식

1. Lab1_1

1-1. 실습 프로그램의 구성 및 동작 원리

n값을 받아서 0부터 n까지 정수를 더하는 프로그램 sum(n)

sum(n)은 n이 0이 아닐 때 n값과 sum(n-1)값을 **재귀**적으로 호출한 값을 더하여 리턴한다.
n이 0일때는 0을 리턴. 리턴할 값은 $s0레지스터에 저장하고있다가 마지막에 $v0으로 넘겨준다

```
sum:
addiu   $sp, $sp, -8            # increase stack size by 8
sw      $ra, 4($sp)             # store saved register ra
sw      $s0, 0($sp)             # store saved register s0
```

필요한 만큼 스택포인터를 증가시켜 saved register($ra, $s0)를 백업한다

```
bne     $a0, $0, notequal       # if they aren't equal, go to notequal
add     $s0, $0, $0             # set value of s0 to zero
j       return                  # go to return
```

n($a0)이 0 과 같으면 0 을 리턴

```
notequal:
add     $s0, $a0, $0            # set value of s0 to a0
addi    $a0, $a0, -1            # decrease value of argument by 1
jal     sum                     # call sum
add     $s0, $s0, $v0           # add returned value to s0
```

n 이 0 이 아니면 n 값과 sum(n-1)값을 더해서 리턴 (재귀적으로 호출)

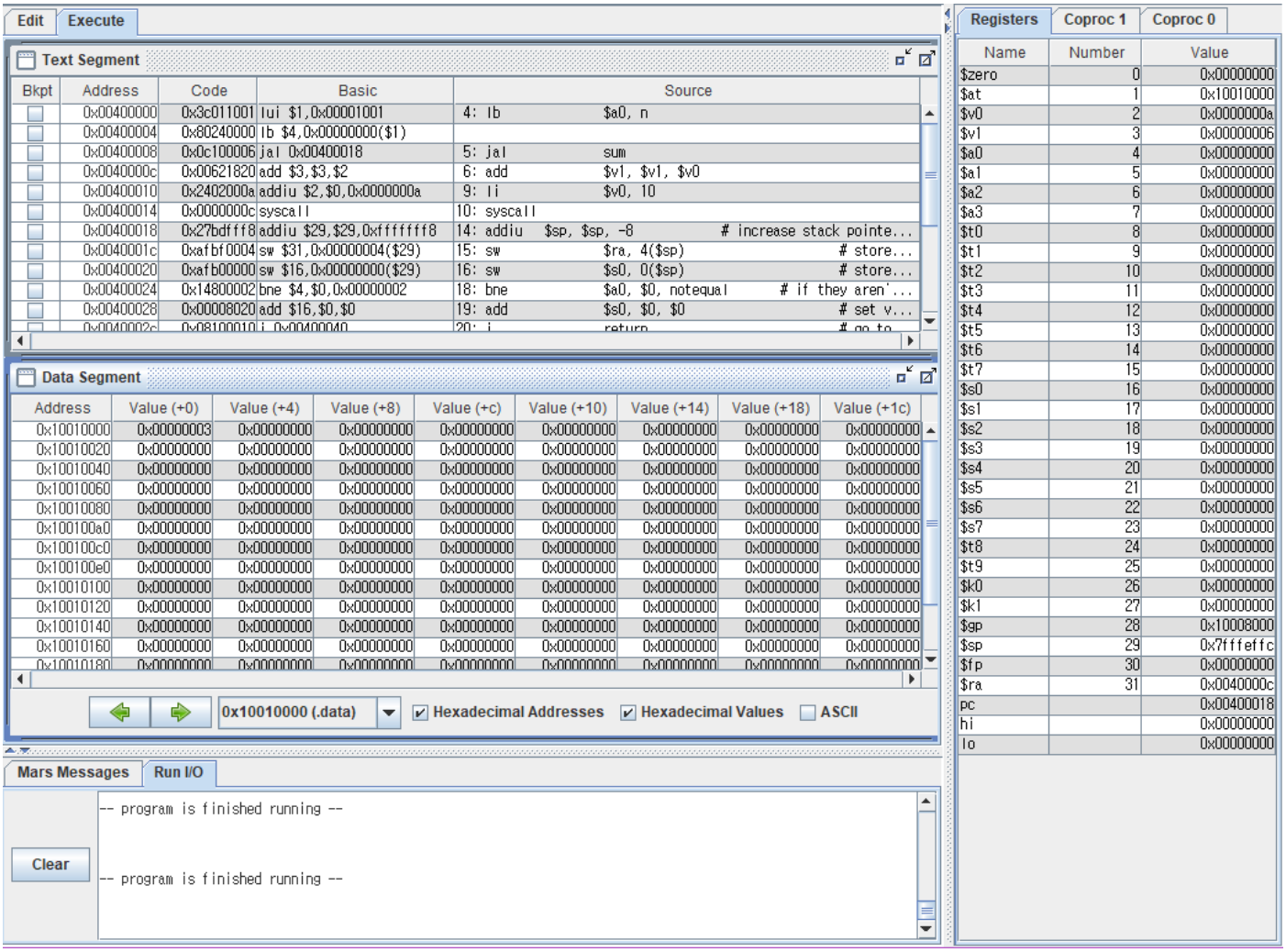
```
return:
add     $v0, $s0, $0            # set return value by s0
lw      $ra, 4($sp)             # load saved register ra
lw      $s0, 0($sp)             # load saved register s0
addiu   $sp, $sp, 8             # decrease stack size by 8
jr      $ra                     # return to caller
```

$s0 을 리턴값으로 설정한다
스택에서 saved register($ra, $s0)를 복원하고 스택포인터를 감소시킨다
Caller 가 호출한 지점으로 돌아간다

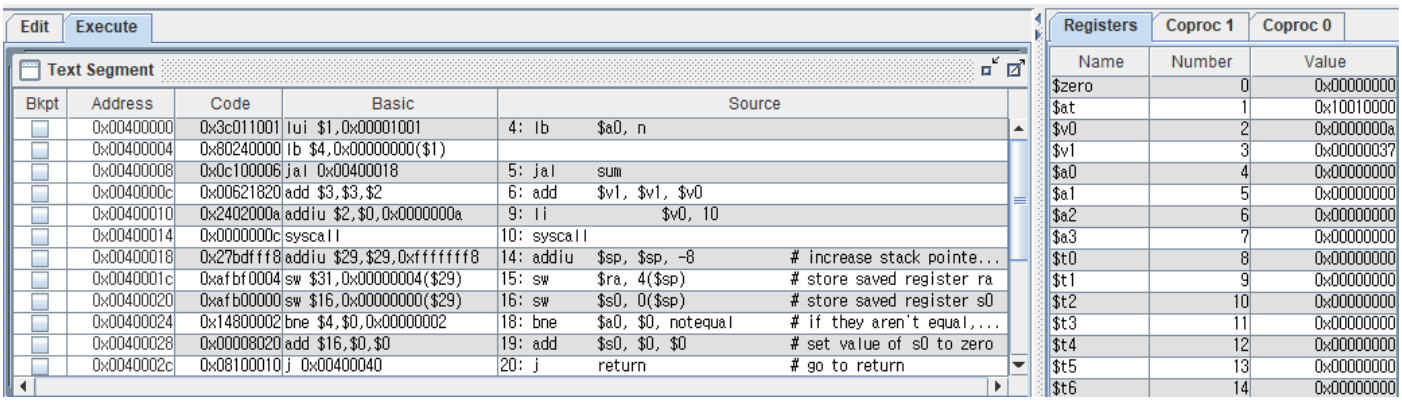
재귀적으로 호출하면서 레지스터의 값을 유지하는 부분이 헷갈렸는데 한줄 단위로 실행하면서 디버깅을 하니까 쉽게 해결할 수 있었다.

1-2. 결과

프로그램이 종료되고 $v1레지스터를 $v0레지스터로 전달받은 값으로 설정하고 $v0레지스터는 0xA으로 설정된 모습 (n=3, 3 + 2 + 1 + 0 = 6)



n이 10일 때 55(0x37)값이 정상적으로 리턴된 모습



1-3. 결론

다른 n값을 넣었을때도 sum(n)을 실행하였을 때 0부터 n까지 정수들을 더한값이 정상적으로 리턴된다.

2. Lab1_2

1-1. 실습 프로그램의 구성 및 동작 원리

n값을 받아서 피보나치 수열의 n번째 값을 구하는 프로그램 fib(n)

fib(n)은 n이 0이면 0을 리턴, 1이면 1을 리턴, 그 외의 경우에는 fib(n-1)값과 fib(n-2)의 값을 **재귀**적으로 호출한 값을 더하여 리턴한다. fib(n-1)의 값을 $t0에 저장한 후에 fib(n-2)의 값과 $t0의 값을 더하여 리턴한다.

```
fib:
bne     $a0, $0, notequal1      # if they aren't equal, go to notequal1
add     $v0, $0, $0             # set return value by zero
j       return                  # go to return
```

n($a0)의 값이 0 과 같으면 0 을 리턴한다

```
notequal1:
bne     $a0, 1, notequal2       # if they aren't equal, go to notequal2
addi    $v0, $0, 1              # set return value by 1
j       return                  # go to return
```

n($a0)의 값이 1 과 같으면 1 을 리턴한다

```
notequal2:
addiu   $sp, $sp, -12           # increase stack size by 12
sw      $ra, 8($sp)             # store saved register ra
sw      $a0, 4($sp)             # store volatile register a0 (argument)
```

스택포인터를 증가시켜 스택에 saved register($ra), volatile register($a0)를 백업한다

```
addi    $a0, $a0, -1            # decrease value of argument by 1
jal     fib                     # call fib
add     $t0, $v0, $0            # set value of t0 to returned value
```

n-1 을 인자로 하여 fib 를 재귀적으로 호출한 값을 $t0 에 저장한다

```
sw      $t0, 0($sp)             # store volatile register t0
lw      $a0, 4($sp)             # load volatile register a0 (argument)
addi    $a0, $a0, -2            # decrease value of argument by 2
jal     fib                     # call fib
```

스택에 volatile register($t0)를 백업하고 volatile register($a0)을 복원한다

n-2 을 인자로 하여 fib 를 재귀적으로 호출한다

```
lw      $t0, 0($sp)             # load volatile register t0
add     $v0, $t0, $v0           # set return value to (t0 + returned value)
lw      $ra, 8($sp)             # load saved register ra
addiu   $sp, $sp, 12            # decrease stack size by 8
```

스택에서 volatile register($t0)을 복원한다

fib(n-2)의 결과값과 $t0 을 더한 값을 리턴값으로 설정한다

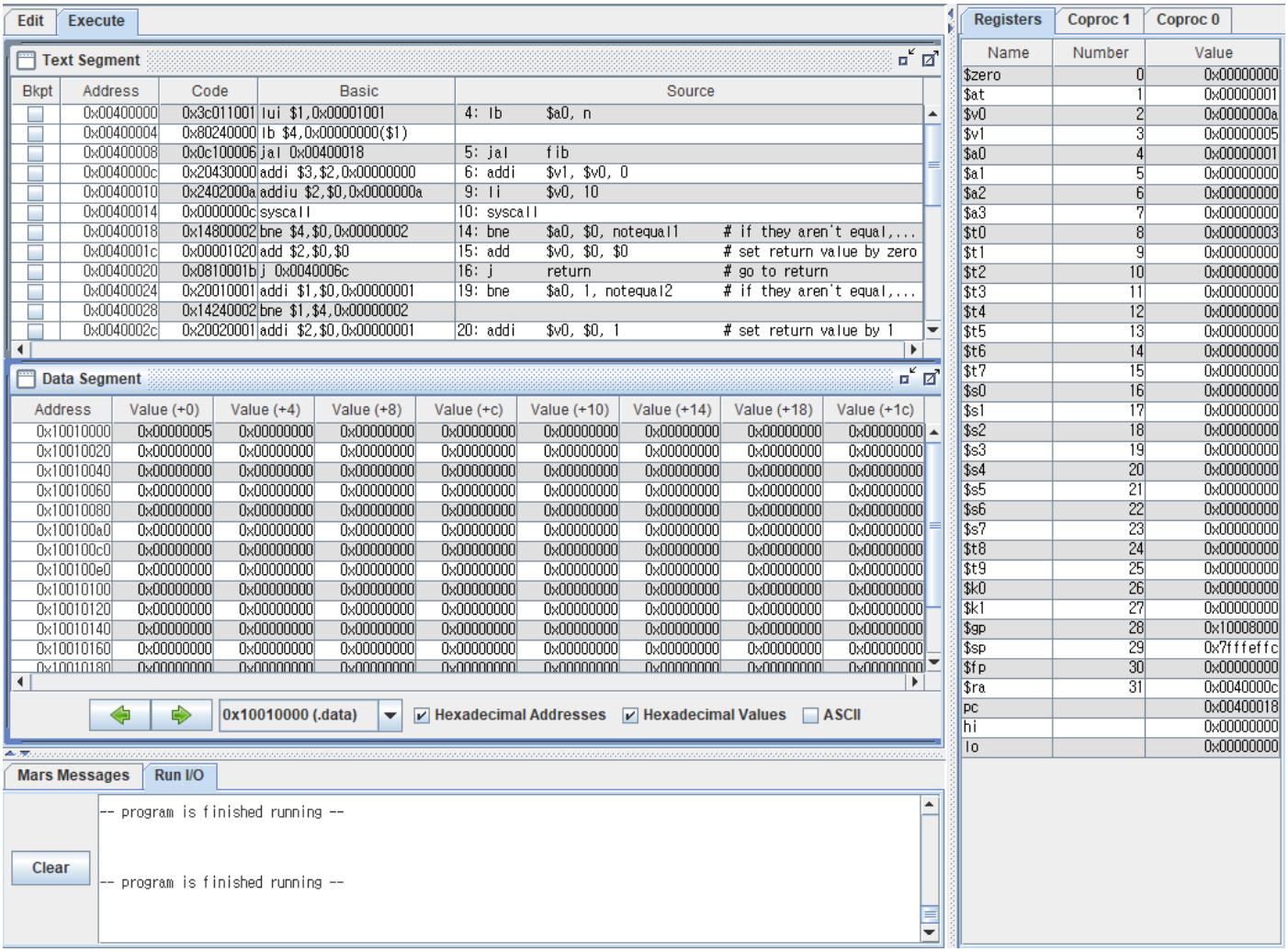
스택에서 saved register($ra)를 복원하고 스택포인터를 감소시킨다

```
return:
jr      $ra                     # return to caller
```

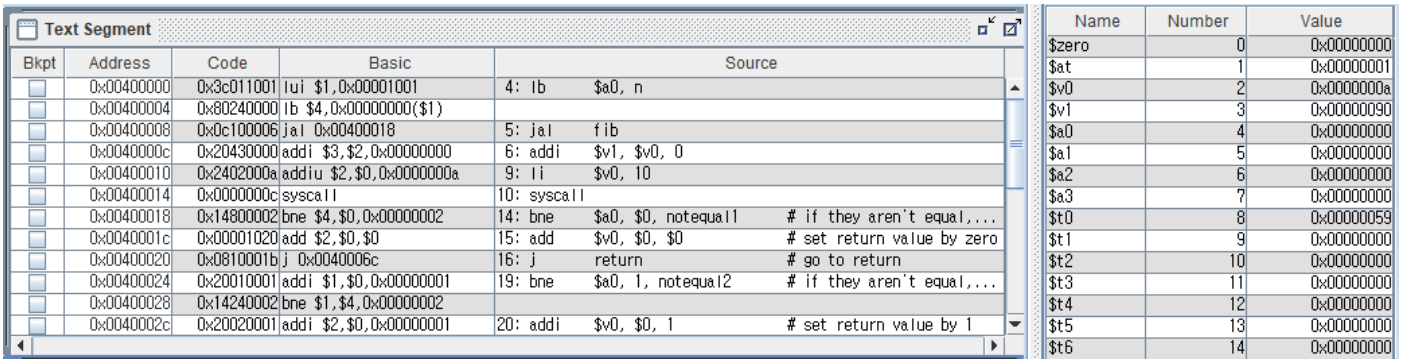
Caller가 호출한 지점으로 돌아간다

1-2. 결과

프로그램이 종료되고 $v1레지스터를 $v0레지스터로 전달받은 값으로 설정하고 $v0레지스터는 0xA으로 설정된 모습 fib(5) = fib(4) + fib(3) = 3 + 2 = 5



n이 12일 때 144(0x90)값이 정상적으로 리턴된 모습 fib(12) = 144



1-3. 결론

다른 n값을 넣었을때도 fib(n)을 실행하였을 때 피보나치 수열의 n번째 값이 정상적으로 리턴된다.