

HW_1

이명규 (201716422)

전북대학교 컴퓨터공학부

이명규 lee95292@naver.com

HW_1-1 - SUM Procedure 요약

n을 인자로 받아 n까지의 합을 꼬리물기 재귀를 통해 더해주는 함수를 작성합니다.
\$ra 레지스터와 jal, 스택의 개념을 사용해 재귀를 구현하였습니다.

1. 실습 프로그램의 구성 및 동작 원리

sum 함수를 mips로 구성하는 과정에서, 세 가지 구현사항이 있었습니다.

```
13 addi $sp, $sp, -8
14 sw $ra, 4($sp)
15 sw $a0, 0($sp) # push n, $ra -> stack
```

1; 현재 함수의 Return Address(\$ra)와 인자값인 n값을 스택에 push해야 합니다. 이 값들은 2번에서 재귀함수가 호출되며 변경되지만, 3번 과정에서 사용되기 때문입니다.

```
16 addi $a0, $a0, -1 # n-1 -> a0
17 beq $a0, $0, then
18 jal sum
```

2: 재귀함수의 boundary condition을 체크합니다.
이 때, n-1이 0일 경우, 즉 n=1일 때 3번과정으로 넘어가지만, 그렇지 않을 경우, n-1을 인자로 설정해, 자기 자신인 sum을 다시 호출합니다.

```
19 then:
20 lw $t1, 0($sp)
21 lw $t2, 4($sp) #pop $ra, n -> $t1, $t2
22 add $v0, $t1, $v0 # ret=n+sum(n-1)
23 addi $sp, $sp, 8 #stack cover
24 jr $ra #return
```

3: 1번과정에서 저장했던, 상위 재귀의 인자값(\$t1=n)과, 현재 함수의 리턴값(\$v0 = sum(n-1))을 Pop해준 후, 더해 리턴(jr \$ra)해줍니다.
리턴 이전에 사용한 스택을 반환해줍니다.

어려웠던 점.

처음 sum함수를 구현할 때, 재귀함수가 mips에서 어떠한 과정을 통해 동작하는지 개념이 정립되지 않았지만, 여러 참고자료들을 통해, mips에서 재귀함수의 동작원리를 파악하여, 성공적으로 구현할 수 있었습니다.

2. 결과

\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000f
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffffc
\$fp	30	0x00000000
\$ra	31	0x00400030
pc		0x00400038
hi		0x00000000
lo		0x00000000

mips구동환경에서 sum.asm을 실행 후의 결과입니다.
아래 그림은 n=5일 때, 최종적인 리턴값을 출력하는 v0 레지스터입니다.

\$v0	2	0x0000000f
------	---	------------

n=5로 설정하였으므로,
v0값이 $1 > 1+2 > 1+2+3 > 1+2+3+4 > 1+2+3+4+5$
로 순차적으로 변화하며 결과적으로 1부터 5까지의 합인 15(0x0000000f)를 나타냅니다.

3. 결론

C언어에서 재귀적인 함수를 구현하는 과정을 MIPS를 통해 작성하는 과정에서 재귀함수의 구현에 어려움을 겪었지만, 재귀함수의 본질을 학습하며 해결할 수 있었습니다.

HW_1-2 – Fibo Procedure 요약

n을 인자로 받아 n 번째 피보나치 수를 리턴하는 함수를 작성합니다.

1-1 sum 함수와 마찬가지로 재귀함수를 통해 값을 구합니다.

하지만, Sum 함수와 다르게, 2개의 base case를 가지며 재귀 또한 2갈래로 나뉘어 진행된다는 차이가 있습니다.

1. 실습 프로그램의 구성 및 동작 원리

fibonacci 함수를 MIPS로 구현하는 과정에서, 세 가지의 구현사항이 있었습니다.

1. Boundary condition 정의
2. 재귀함수 호출
3. return 과정에서 피보나치 수 구하기.

아래 코드는 구현사항에 대한 MIPS 코드와 부연설명입니다.

```
12 fib:
13 addi $sp, $sp, -12
14 sw $s0, 8($sp)      #Push: s0 = n for new arg: n-1 or n-2
15 sw $s1, 4($sp)      #s1 = return value
16 sw $ra, 0($sp)      #$ra= ret3
17
18 slti $t0, $a0, 2
19 beq $t0, $0, then    #if n>=2
20 add $v0, $0, $a0     #if n=1, n=0
21 j end
```

1-1: 13~16에서는 2번 과정의 재귀에서 사용될 값들인 Return address, return value, argument 값들을 스택에 Push해줍니다.

이 세 개의 값들은, 재귀함수의 호출과정 또는 호출 이후에도 사용해야하기 때문에 따로 저장합니다.

1-2: 18~21라인에서는 Boundary Condition을 체크합니다. n=1, n=0일 경우 체크되며, 이 경우에는 재귀함수를 호출하지 않습니다.

```
23 then:  #fib(n-1)+fib(n-2)
24 addi $s0, $a0, 0
25 addi $a0, $a0, -1      #fib(n-1)
26 jal fib
27
28 addi $s1, $v0, 0
29 addi $a0, $s0, -2      #fib(n-2)
30 jal fib
31
32 add $v0, $s1, $v0
```

2: then Label에서는 , Boundary Condition에서 체크되지 않은 경우 fib(n-1)+fib(n-2)를 연산합니다. 이 과정에서

1. \$a0 레지스터를 각각 n-1, n-2로 변경하며, (25,29)
2. 현재 실행중인 함수의 인자값 n을 저장(24)하고
3. fibo(n-2)를 연산하는 경우에는 이미 완료된 연산인 fibo(n-1)값을 \$s1에 저장합니다.(28)

이렇게 fibo(n-1)과 fibo(n-2)연산이 모두 끝나면, 두 값을 더해준 뒤, end Label로 이동합니다.

```
34 end:
35 lw $ra, 0($sp)      #pop
36 lw $s1, 4($sp)
37 lw $s0, 8($sp)
38 addi $sp, $sp, 12
39 jr $ra
```

3: 세 번째 과정에서는 재귀로 호출된 함수가 종료되는 프로세스를 다룹니다.

end 과정에서, fibo 함수를 사용하기 위해 Push했던 \$s1 : 임시 fibo(n-1) 저장 레지스터
, \$s0 : 현재 인자값 저장 레지스터
\$ra : 현재 함수의 return address 들을 이전 함수에서

저장했던 값으로 불러오기 위해 stack에서 Pop해주고,
스택을 사용한만큼 반환한 뒤, 리턴합니다.

2. 결과

\$zero	0	0.00000000
\$at	1	0.10010000
\$v0	2	0.00000000
\$v1	3	0.00000005
\$a0	4	0.00000001
\$a1	5	0.00000000
\$a2	6	0.00000000
\$a3	7	0.00000000
\$t0	8	0.00000001
\$t1	9	0.00000000
\$t2	10	0.00000000
\$t3	11	0.00000000
\$t4	12	0.00000000
\$t5	13	0.00000000
\$t6	14	0.00000000
\$t7	15	0.00000000
\$t8	16	0.00000000
\$t9	17	0.00000000
\$s2	18	0.00000000
\$s3	19	0.00000000
\$s4	20	0.00000000
\$s5	21	0.00000000
\$s6	22	0.00000000
\$s7	23	0.00000000
\$s8	24	0.00000000
\$s9	25	0.00000000
\$s0	26	0.00000000
\$k1	27	0.00000000
\$s0	28	0.10010000
\$s0	29	0.71111111
\$f0	30	0.00000000
\$ra	31	0.00400000
pc		0.00400018

mips구동환경에서 fibo.asm을 실행 후의 결과입니다.

아래 그림은 n=5일 때, 최종적인 리턴값을 출력하는 v0 레지스터입니다.



\$v1 레지스터에 값 5가 저장되어있습니다.

fibo 함수는 재귀적 과정을 통해 약 32회(2의 5승)
호출되며, 이 과정에는 상당한 중복이 있어
비효율적이지만,

fibo(1) =1 부터 bottom- up과정을 통해
0> 1 > 1 > 2 > 3 > 5 의 값을 도출하게 됩니다.

3. 결론

구현과정에서 fibo(n-1)+ fibo(n-2)연산을 수행할 때,
원하지 않는 값이 계속 나와 원인을 찾지 못했습니다.
이 때 재귀가 두 갈래로 나뉘는 경우, return 값을
저장하는 \$v0값을 스택에 저장해 연산해주어야 한다는
것을 깨닫고, \$s1레지스터를 도입해 값을 도출해낼 수
있었습니다.