

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 **n**까지 정수를 더하는 프로그램 **sum(n)**

sum(n)은 **n**이 0이 아닐 때 **n**값과 sum(n-1)값을 재귀적으로 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 \$v1 레지스터를 \$v0 레지스터로 전달받은 값으로 설정하고 \$v0 레지스터는 0xA로 설정된 모습 ($n=3, 3 + 2 + 1 + 0 = 6$)

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x0000000a
\$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
\$s8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x0040000c
pc		0x00400018
hi		0x00000000
lo		0x00000000

n이 10일 때 55(0x37)값이 정상적으로 리턴된 모습

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x00000037
\$v1	3	0x00000037
\$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

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 **n**번째 값을 구하는 프로그램 **fib(n)**

fib(n)은 **n**이 0이면 0을 리턴, 1이면 1을 리턴, 그 외의 경우에는 fib(n-1)값과 fib(n-2)의 값을 **재귀적으로** 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 $\text{fib}(5) = \text{fib}(4) + \text{fib}(3) = 3 + 2 = 5$

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x0001001	4: lb \$a0, n
<input type="checkbox"/>	0x00400004	0x80240000	lb \$4,0x00000000(\$1)	
<input type="checkbox"/>	0x00400008	0x0c100006	jal 0x00400018	5: jal fib
<input type="checkbox"/>	0x0040000c	0x20430000	addi \$3,\$2,0x00000000	6: addi \$v1, \$v0, 0
<input type="checkbox"/>	0x00400010	0x2402000a	addiu \$2,\$0,0x0000000a	9: li \$v0, 10
<input type="checkbox"/>	0x00400014	0x0000000c	syscall	10: syscall
<input type="checkbox"/>	0x00400018	0x14800002	bne \$4,\$0,0x00000002	14: bne \$a0, \$0, notequal1 # if they aren't equal,...
<input type="checkbox"/>	0x0040001c	0x00001020	add \$2,\$0,\$0	15: add \$v0, \$0, \$0 # set return value by zero
<input type="checkbox"/>	0x00400020	0x0810001b	j 0x0040006c	16: j return # go to return
<input type="checkbox"/>	0x00400024	0x20010001	addi \$1,\$0,0x00000001	19: bne \$a0, 1, notequal2 # if they aren't equal,...
<input type="checkbox"/>	0x00400028	0x14240002	bne \$1,\$4,0x00000002	
<input type="checkbox"/>	0x0040002c	0x20020001	addi \$2,\$0,0x00000001	20: addi \$v0, \$0, 1 # set return value by 1

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000005	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data)

☒ Hexadecimal Addresses
 ☒ Hexadecimal Values
 ☐ ASCII

Mars Messages Run I/O

Clear

```
-- program is finished running --

-- program is finished running --
```

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000001
\$v0	2	0x0000000a
\$v1	3	0x00000005
\$a0	4	0x00000001
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000003
\$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	0x7fffffc0
\$fp	30	0x00000000
\$ra	31	0x0040000c
pc		0x00400018
hi		0x00000000
lo		0x00000000

n이 12일 때 144(0x90)값이 정상적으로 리턴된 모습 $\text{fib}(12) = 144$

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x0001001	4: lb \$a0, n
<input type="checkbox"/>	0x00400004	0x80240000	lb \$4,0x00000000(\$1)	
<input type="checkbox"/>	0x00400008	0x0c100006	jal 0x00400018	5: jal fib
<input type="checkbox"/>	0x0040000c	0x20430000	addi \$3,\$2,0x00000000	6: addi \$v1, \$v0, 0
<input type="checkbox"/>	0x00400010	0x2402000a	addiu \$2,\$0,0x0000000a	9: li \$v0, 10
<input type="checkbox"/>	0x00400014	0x0000000c	syscall	10: syscall
<input type="checkbox"/>	0x00400018	0x14800002	bne \$4,\$0,0x00000002	14: bne \$a0, \$0, notequal1 # if they aren't equal,...
<input type="checkbox"/>	0x0040001c	0x00001020	add \$2,\$0,\$0	15: add \$v0, \$0, \$0 # set return value by zero
<input type="checkbox"/>	0x00400020	0x0810001b	j 0x0040006c	16: j return # go to return
<input type="checkbox"/>	0x00400024	0x20010001	addi \$1,\$0,0x00000001	19: bne \$a0, 1, notequal2 # if they aren't equal,...
<input type="checkbox"/>	0x00400028	0x14240002	bne \$1,\$4,0x00000002	
<input type="checkbox"/>	0x0040002c	0x20020001	addi \$2,\$0,0x00000001	20: addi \$v0, \$0, 1 # set return value by 1

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000001
\$v0	2	0x0000000a
\$v1	3	0x00000090
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000059
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000

1-3. 결론

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