Computer Architecture (ENE1004)

▼ Lec 6

Lec 6: Instructions: Language of the Computer 5

Nested Procedures (중첩 프로시저)

- All procedures are not leaf procedures, which do not call others
  - main() calls func_A(), which calls func_B(); here, func_A() is a nested procedure
  - Recursive procedures are also nested
- A problematic situation in nested procedures
  - main() calls procedure A with an argument of 3 (① addi $a0, $zero, 3; ② jal A)
  - Procedure A calls procedure B with an argument of 7 (③ addi $a0, $zero, 7; ④ jal B)
  - You may find two conflicts;
    - At ③, procedure B updates $a0 with 7; what if procedure A continues to expect that $a0 holds 3? : ③에서 프로시저
    - At ④, procedure B updates $ra with its return address; procedure A loses its return address
- One solution is to push all the registers that must be preserved onto the stack
  - Caller pushes arg registers ($a0-$a3) or temp registers ($t0-$t9) that are needed after the call
  - Callee pushes return address register ($ra) and saved registers ($s0-$s7) used by the callee
  - Note that stack pointer ($sp) should be adjusted correspondingly
- 모든 프로시저는 다른 프로시저를 호출하지 않는 leaf 프로시저가 아니다.

    - main()은 func_A()를 호출하고, 이 함수는 func_B()를 호출한다. 여기서 func_A()는 중첩된 프로시저이다.
    - 재귀 프로시저도 중첩된다.
- 중첩 프로시저에서 문제가 되는 상황
    - main()은 인수가 3인 프로시저 A를 호출한다. (① addi $a0, $zero, 3; ② jal A)
    - 프로시저 A는 인수가 7인 프로시저 B를 호출한다. (③ addi $a0, $zero, 7; ④ jal B)
    - 두 가지 충돌이 발생할 수 있다.
        - ③에서 프로시저 B는 $a0을 7로 업데이트하는데, 만약 프로시저 A가 계속해서 $a0이 3을 보유할 것으로 예상한다면 어떻게 될까?
        - ④에서 프로시저 B는 $ra를 반환 주소로 업데이트하고, 프로시저 A는 반환 주소를 잃는다.
- 한 가지 해결책은 보존해야 하는 모든 레지스터를 스택에 푸시하는 것이다.
    - 호출자는 호출 후에 필요한 인자 레지스터 ($a0-$a3) 또는 ($t0-$t9)를 푸시한다.
    - 수신자는 수신자가 사용한 반환 주소 레지스터($ra)와 저장된 레지스터 ($s0-$s7)를 푸시한다.
    - 스택 포인터 $sp는 그에 따라 조정되어야 한다.
- 왜 호출자는 $t0-$t9를 푸시하고, 왜 수신자는 $s0-$s7를 푸시하는가?
    - $t0-$t9 레지스터는 임시로 사용되는 레지스터이다. 이러한 레지스터들은 함수 호출 이후에도 그 값이 유지할 필요가 없는 임시 데이터에 사용된다. 따라서 호출자는 함수 호출 전에 호출 이후에 필요한 임시 데이터를 이 레지스터들에 저장하고, 이후에 다시 사용할 수 있도록 스택에 저장한다. (중첩 프로시저이기 때문에 임시 레지스터가 필요할 수 있음.)
    - $s0-$s7 레지스터는 함수가 호출되고 반환되는 동안 값이 유지되어야 하는 레지스터이다. 이러한 레지스터들은 함수 호출 이후에도 그 값을 유지해야 하므로, 수신자가 이러한 레지스터들을 스택에 저장하여 나중에 복원할 수 있도록 한다.

Nested Procedures: Example

```
int fact (int n)
{
    if (n < 1)
        return (1);
    else
        return (n * fact(n-1));
}
```

is translated into

```
fact:

addi $sp, $sp, -8 # adjust stack for 2 items
sw $ra, 4($sp) # save the return address
sw $a0, 0($sp) # save the argument n

slti $t0, $a0, 1 # test for n < 1
beq $t0, $zero, L1 # if n >= 1, go to L1

addi $v0, $zero, 1 # return 1
addi $sp, $sp, 8 # pop 2 items off stack
jr $ra # return to caller

L1:
addi $a0, $a0, -1 # n >= 1 : arg gets (n-1)
jal fact # call fact with (n-1)

lw $a0, 0($sp) # return from jal; restore arg n
lw $ra, 4($sp) # restore return address
addi $sp, $sp, 8 # adjust $sp to pop 2 items

mul $v0, $a0, $v0 # return n * fact(n-1)
jr $ra # return to caller
```

- `addi $sp, $sp, -8 # adjust stack for 2 items`
  `sw $ra, 4($sp) # save the return address`
  `sw $a0, 0($sp) # save the argument n`

$a0 and $ra can be used in the subsequent call, which is kept onto the stack (후속 호출에서 $a0 및 $ra를 사용할 수 있으며, 스택에 유지된다.)

- `slti $t0, $a0, 1 # test for n < 1`
  `beq $t0, #zero, L1`

  slti & beq for if-then-else statement

- `addi $v0, $zero, 1 # return 1`
  `addi $sp, $sp, 8 # pop 2 items off stack`
  `jr $ra # return to caller`

  if n < 1, this leaf procedure returns to the caller; here, $a0 and $ra still hole the original values; so, you don't have to get those values from the stack (n < 1이면 이 리프 프로시저는 호출자에게 반환된다. 여기서, $a0과 $ra는 여전히 원래 값을 유지한다. 마지막 스택에서 해당 값을 가져올 필요가 없다.)

- `L1:`
  `addi $a0, $a0, -1 # n >= 1 : arg gets (n-1)`
  `jal fact # call fact with (n-1)`

  if n ≥ 1, fact(n-1) is called

- `lw $a0, 0($sp) # return from jal; restore arg n`
  `lw $ra, 4($sp) # restore return address`
  `addi $sp, $sp, 8 # adjust $sp to pop 2 items`

  The return address of fact() is here $a0 and $ra are restored, and $sp is readjusted (fact()의 반환 주소는 여기에 $a0 및 $ra가 복원되고 $sp가 재조정된다.)

- `mul $v0, $a0, $v0 # return n * fact(n-1)`
  `jr $ra # return to caller`

  The current routine returns to the caller with an argument of n * fact(n-1) (현재 루틴은 호출자에게 n * fact(n-1)의 인수를 반환한다.)

- Example: fact (2)

```
fact: # fact(2)

addi $sp, $sp, -8 # adjust stack for 2 items
sw $ra, 4($sp) # fact(2)의 반환 주소
sw $a0, 0($sp) # save the n = 2

slti $t0, $a0, 1 # $t0 = 0
beq $t0, $zero, L1 # $t0 = 0, go to L1
```

```
L1:
addi $a0, $a0, -1 # n = 1
jal fact # call fact(1)

    fact: # fact(1)

    addi $sp, $sp, -8 # adjust stack for 2 items
    sw $ra, 4($sp) # fact(1)의 반환 주소
    sw $a0, 0($sp) # save the n = 1

    slti $t0, $a0, 1 # $t0 = 0
    beq $t0, $zero, L1 # $t0 = 0, go to L1

    L1:
    addi $a0, $a0, -1 # n = 0
    jal fact # call fact(0)

        fact: # fact(0)

        addi $sp, $sp, -8 # adjust stack for 2 items
        sw $ra, 4($sp) # save the fact(0)의 반환 주소
        sw $a0, 0($sp) # save the n = 0

        slti $t0, $a0, 1 # $t0 = 1
        beq $t0, $zero, L1 # $t0 = 1, don't go L1

        addi $v0, $zero, 1 # return 1, $v0 = fact(0)
        addi $sp, $sp, 8 # pop 2 items off stack
        jr $ra # return to caller

    lw $a0, 0($sp) # $a0 = 1
    lw $ra, 4($sp) # $ra = fact(1)의 반환 주소
    addi $sp, $sp, 8 # adjust $sp to pop 2 items

    mul $v0, $a0, $v0 # fact(1) = 1 * fact(0)
    jr $ra # return to caller
```

```
lw $a0, 0($sp) # $a0 = 2
lw $ra, 4($sp) # $ra = fact(2)의 반환 주소
addi $sp, $sp, 8 # adjust $sp to pop 2 items

mul $v0, $a0, $v0 # fact(2) = 2 * fact(1)
jr $ra # return to caller
```