Computer Systems
Lecture 15

Overview

- Stack frame
- Recursive subroutine
- Recursive method for factorial function in Java
- Implementation of recursive function in the assembly language

Stack frame

- The area of the stack which holds all the data related to one call of a subroutine.
- The data includes:
  - Parameters of the subroutine.
  - Return address.
  - Old stack pointer contents (EBP)
  - Local variables.

Summary: a scenario of nested subroutine calls

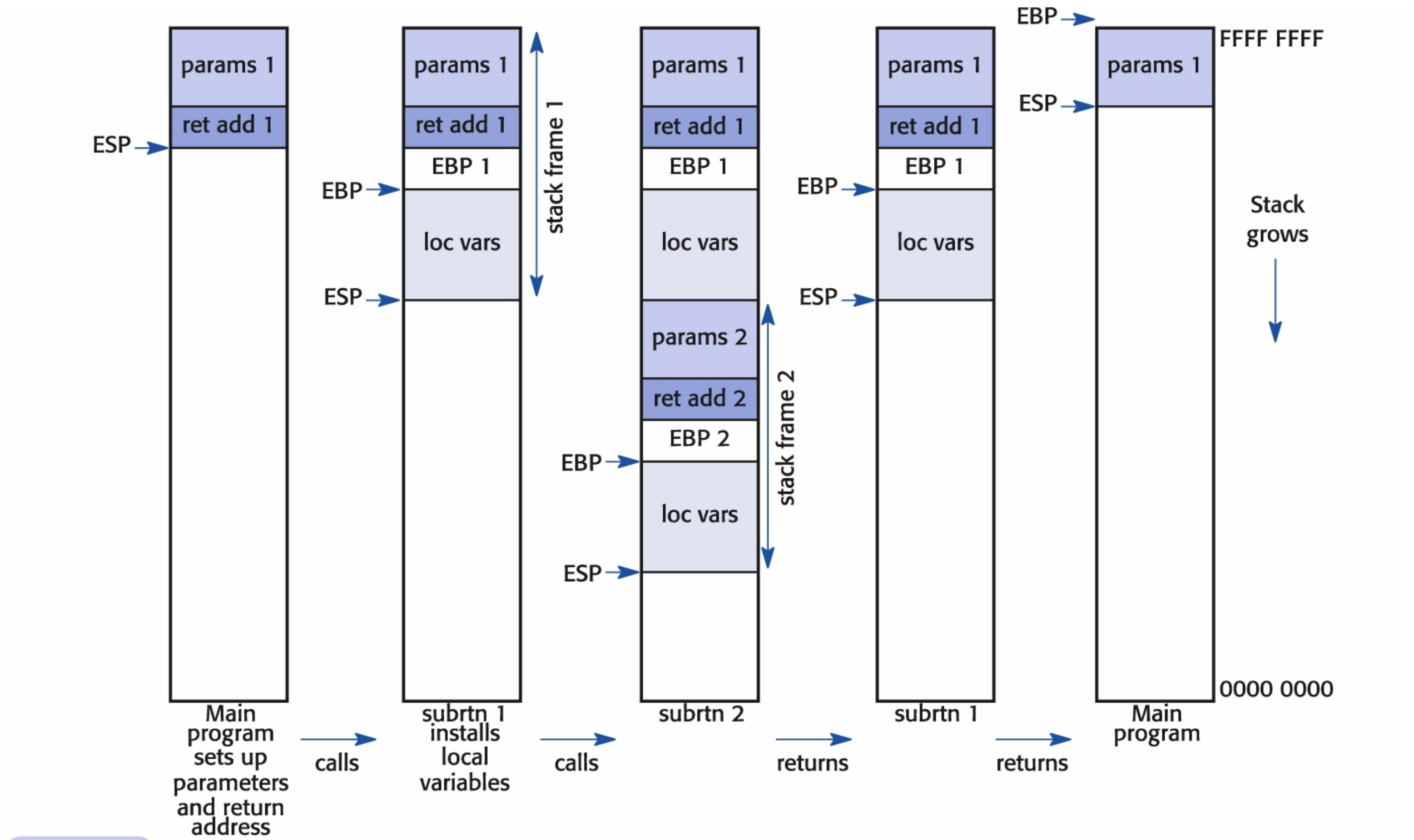


Fig. 8.9 Using thestack for local variables.

Recursive subroutines

- A recursive subroutine, or procedure, is one that may in some circumstances calls **itself** to perform some subsidiary task.
- For example a subroutine `SUBR` may `CALL SUBR`.
- Recursion may appear in a more subtle form of **mutual recursion**, when, e.g.,
  - `SUBR1` calls `SUBR2`, and
  - `SUBR2` in turn calls `SUBR1`.

Examples of recursive definitions (procedures)

- Factorial function.
  - – factorial(1) = 1.
  - – factorial(n) = n * factorial(n-1).
- Merge sort.
  - – given a list, split it into two parts
  - – apply *Merge sort* to each part
  - – merge the results.

Recursive method for factorial function in Java

```
…
static long factorial(int n)
{
   if (n < 2) return 1;
   return n * factorial(n-1);
}
…
```

Implementation in the assembly language

- One auxiliary procedure to be used in the recursive `factorial` procedure:

```
multiply PROC          // input from two top values on the stack
        pop eax        // pop a value from the stack to eax
        mov aux, eax   // move this value to the auxiliary variable
        pop eax        // pop one more value from the stack
        mul eax, aux   // multiply these two values
        ret            // return the result in eax
multiply ENDP
```

- It takes two top values on the stack, multiplies them and return the result in `eax` register.
- Side effect?
  - Stack modified (2 pops)
  - `eax` original contents replaced with the product

Stacks for recursion: main procedure for the factorial

```
  factorial PROC   //input n in eax
     push eax    // push current value onto the stack
     dec eax     // decrease the value of n
     jz finish   // if it is zero go to finish
     call factorial  // otherwise call factorial
     push eax        // push the result of last
                     // factorial's call to the stack
     call multiply   // call multiply subroutine
     ret             // return
finish: pop eax      // pop the parameter from the
                     // stack into eax
     ret             // return with the result in eax
  factorial ENDP     // side effect?
```

Comments on the procedure

- Parameter *n* is passed to **`factorial`** subroutine via register **`eax`**.
- The register **`eax`** is used also to return the result.

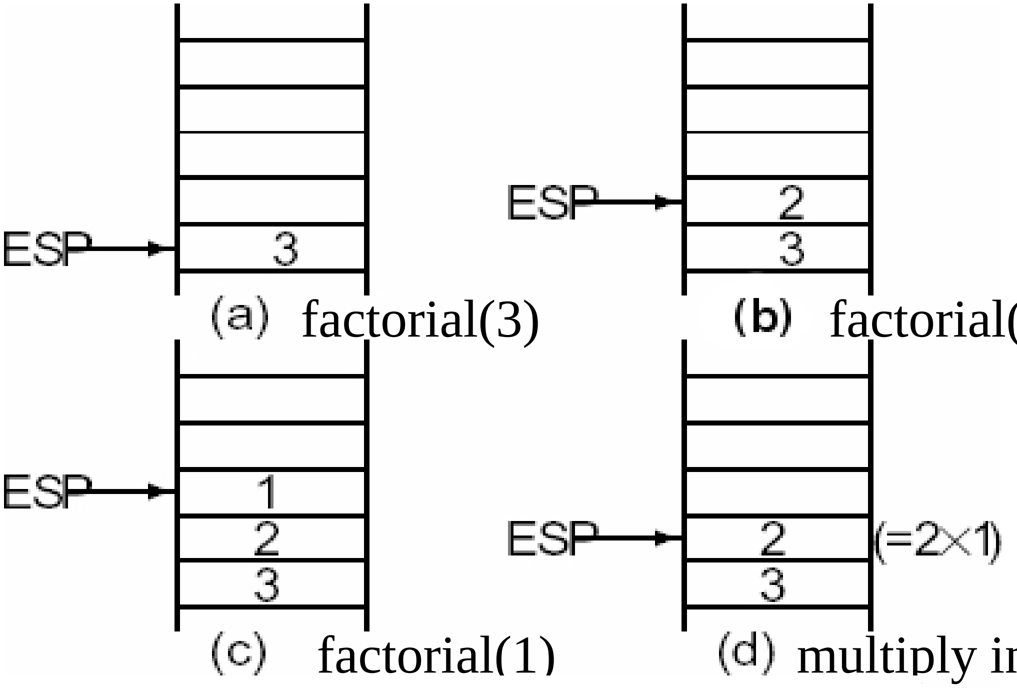
- The 2 parameters to **`multiply`** subroutine are passed via stack.
- The result of **`multiply`** is also returned via **`eax`**.

Computing the factorial of 3

- Consider computing the factorial of 3.
- Calling sequence:

```
        mov eax,3
        call factorial
```

Changing of the stack



(a) factorial(3)

(b) factorial(2)

(c) factorial(1)

(d) multiply in factorial(2)

- Q. A recursive procedure will typically provide an exiting condition. (T or F)

- Q. A recursive procedure will typically have a divide-and-conquer step. (T or F)

- Q. What is the side-effect of the procedure ‘multiply’?

- Q. A recursive procedure can always be re-implemented using iteration without recursion. (T or F)

Readings

- [Wil06] Chapter 8, sections 8.5, 8.6.