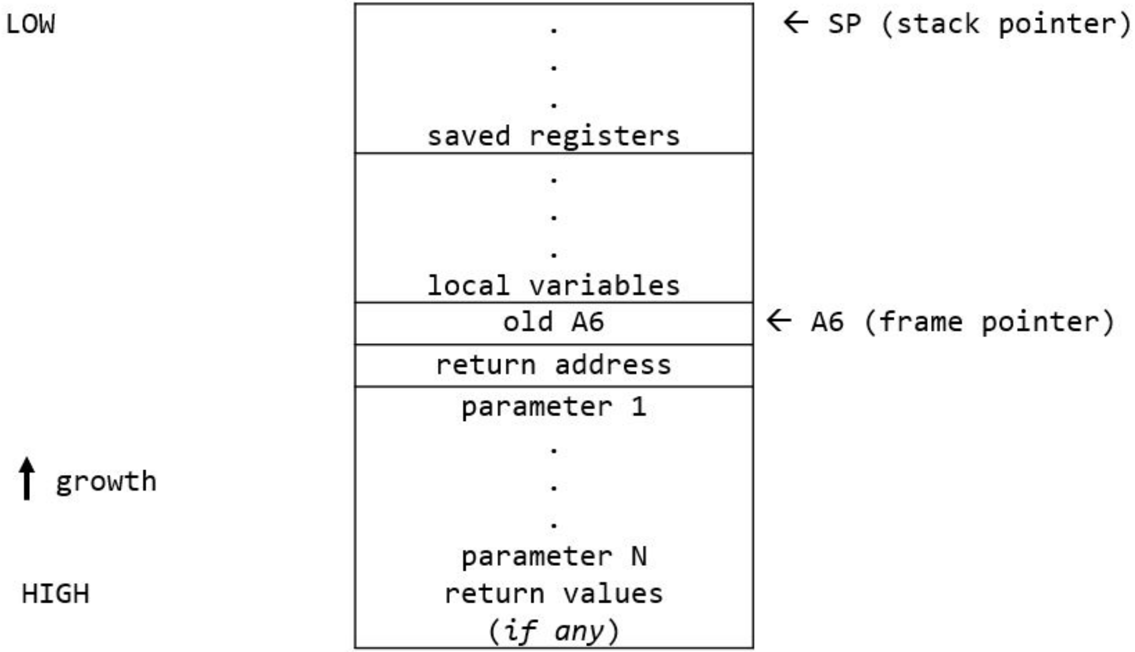
Recursion

Textbook Sections 10.2 - 10.4

Stack Frames Review



LOW

.
.
.
saved registers ← SP (stack pointer)

.
.
.
local variables

old A6 ← A6 (frame pointer)

return address

parameter 1
.
.
.
parameter N

↑ growth

HIGH

return values
(*if any*)

Recursive Subroutines

- A subroutine is recursive if it calls itself
    - E.g. the factorial of n (where n ≥ 0), written n!, is defined by:

```
0! = 1

n! = n × (n-1)! where n ≥ 1
```

Factorial

```
unsigned int fact(unsigned int n)
{
    unsigned int answer;
    if (n == 0)
        answer = 1;             // base case
    else
        answer = n * fact(n - 1);   // recursive case
    return answer;
}
```

Reentrant Subroutines

- A subroutine is reentrant if more than one simultaneous invocation is possible
- All recursive subroutines are reentrant
  - However, a reentrant subroutine does NOT have to be recursive
- Conditions for reentrance:
  - the subroutine code must not be self-modifying
  - when switching between instances, working environment registers must be saved (and later, restored)
  - each instance must have its own stack frame
  - global data must not be used (or access must be “synchronized”)

Tail Recursion

- A recursive subroutine in which the last instruction is the recursive call
    - Other than the **rts** instruction
- Allows greater depth (on stack) of function calls
    - All local variables deallocated
    - All registers restored --- Why does this work?
    - All parameters on the stack

Factorial - Tail Recursion Version

```
unsigned int fact(unsigned int n) {
    return factTR(n, 1);
}

unsigned factTR(unsigned int n, unsigned int a) {
    if (n <= 1)
        return a;                   // base case

    return factTR(n - 1, n * a);// recursive case
}
```