Procedure-Calling Conventions

October 30

CSC201 Section 002

Fall, 2000

Saving registers

- Registers are inevitably used by subroutines; changes their value!
- Registers have global scope; calling procedures also use their contents
  - Conclusion: somebody needs to save registers before the subroutine executes, and restore values afterwards

Copyright 2000, Douglas Reeves

Approaches

- "Caller saves" model
  - caller saves/restores just the registers that it knows have important values
- "Callee saves" model
  - callee saves just the registers that it knows it will modify
- "Hybrid" model
  - callee saves / restores the registers it modifies that are "important" to the caller
- Our convention: "callee saves" model

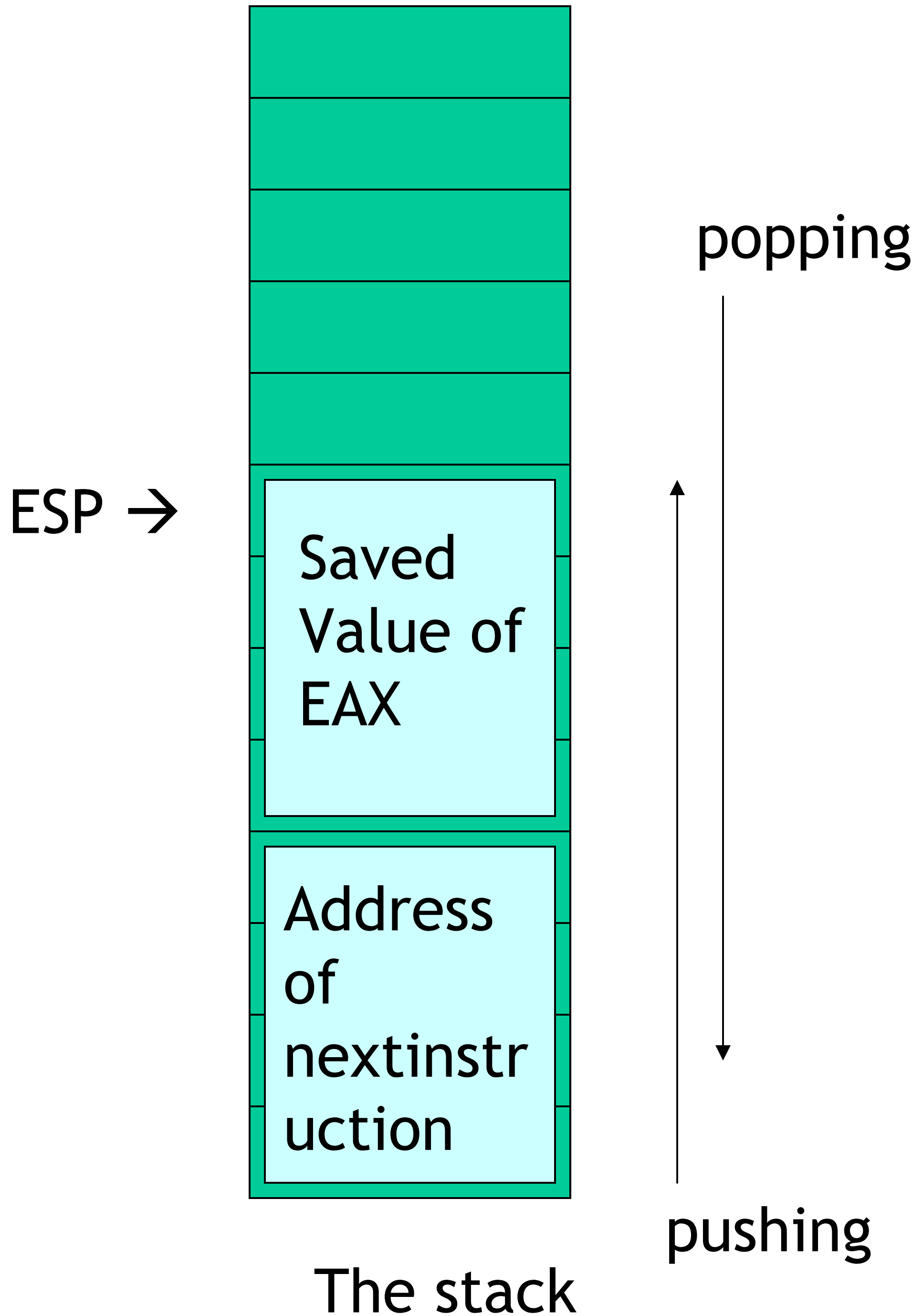
Copyright 2000, Douglas Reeves

Example of Callee Saves

```
main:
        ....
        call    upcase
nextinstruction:
        ...

upcase proc
        push    EAX     ; save EAX value
        get_ch
        add     AL, 23
        put_ch
        pop     EAX     ; restore EAX value
        ret
upcase endp
```


Copyright 2000, Douglas Reeves

Just after the "push EAX" and before the "get_ch"



Copyright 2000, Douglas Reeves

Passing Input Parameters and Return Values

- Registers are global variables; not a great idea to use for this purpose
  - Better choice: the stack
- Before calling the procedure…
  - allocate room on the stack for the return value
  - Push a copy of the input parameter values on the stack
- Some parameters are modified by the procedure as part of its output
  - In this case, push the parameter *address* instead of its value
  - Procedure can modify the parameter using indirect addressing

Copyright 2000, Douglas Reeves

Parameter Passing (cont.)

- "Callee" reads parameter values on stack, using based-displacement addressing
  - EBP = based register
- Before using EBP, must save its value and restore it on exit!

Copyright 2000, Douglas Reeves

"Cleaning Up" the Stack

- After calling a procedure and returning, the stack should look just like it did before the call
- The "callee" must…
  1. Deallocate memory it allocated
  2. Pop saved register values from the stack
  3. Pop the return address
- What about input parameters and the return value?
  - "callee" can deallocate input parameters with "ret x" statement
    - "x" = # of bytes to deallocate (i.e., value to add to ESP)
  - only "caller" can deallocate the return value, after copying

Copyright 2000, Douglas Reeves

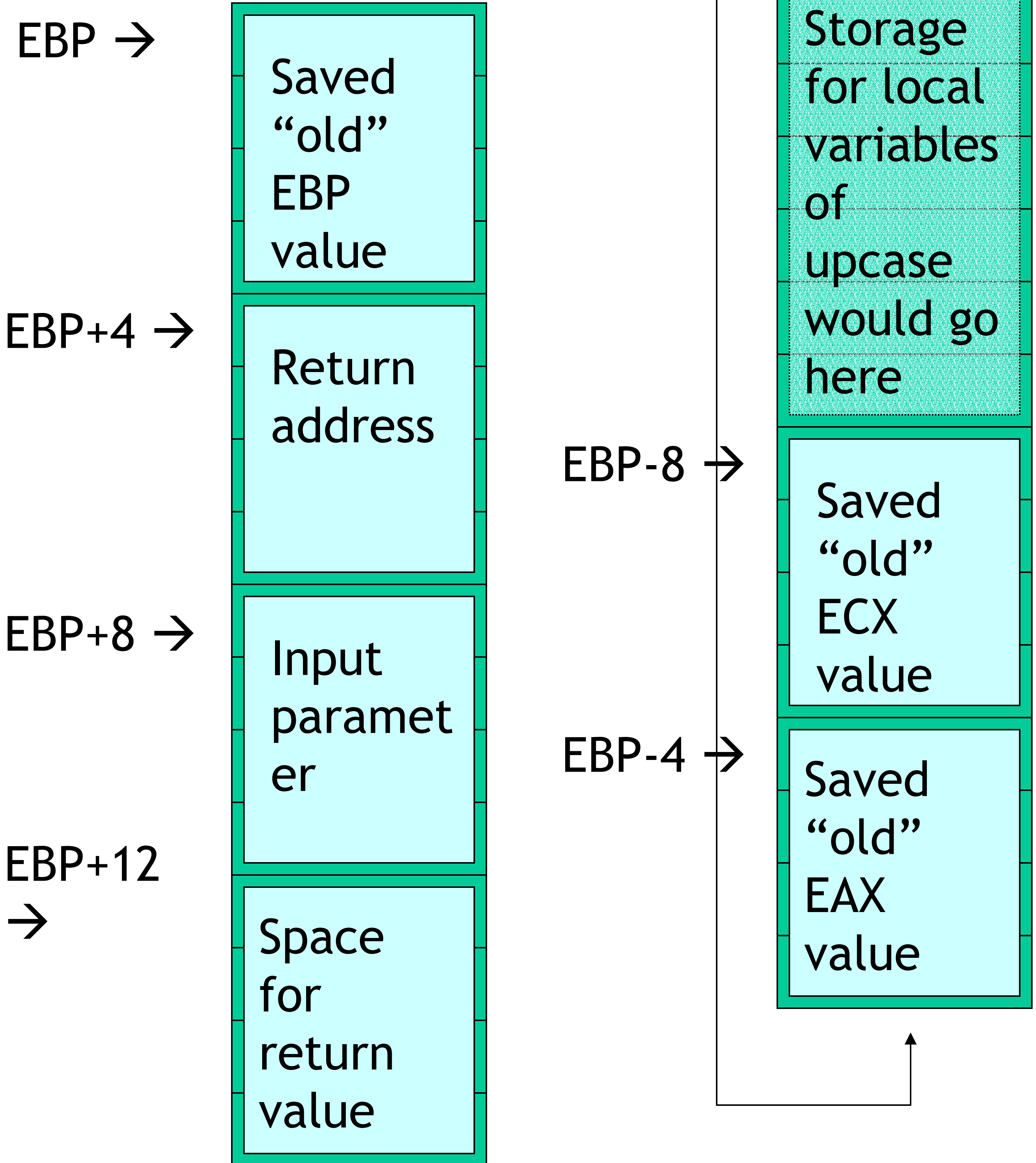
Example

Specification for a procedure:

Write a procedure to read up to n characters from the standard input. Each alphabetic character is mapped to upper case and ouput. Stop when anything other than a lower-case alphabetic character is read, and return to the caller the number of lower case characters read and mapped to upper case.

Copyright 2000, Douglas Reeves

Activation record for procedure *upcase*



The stack

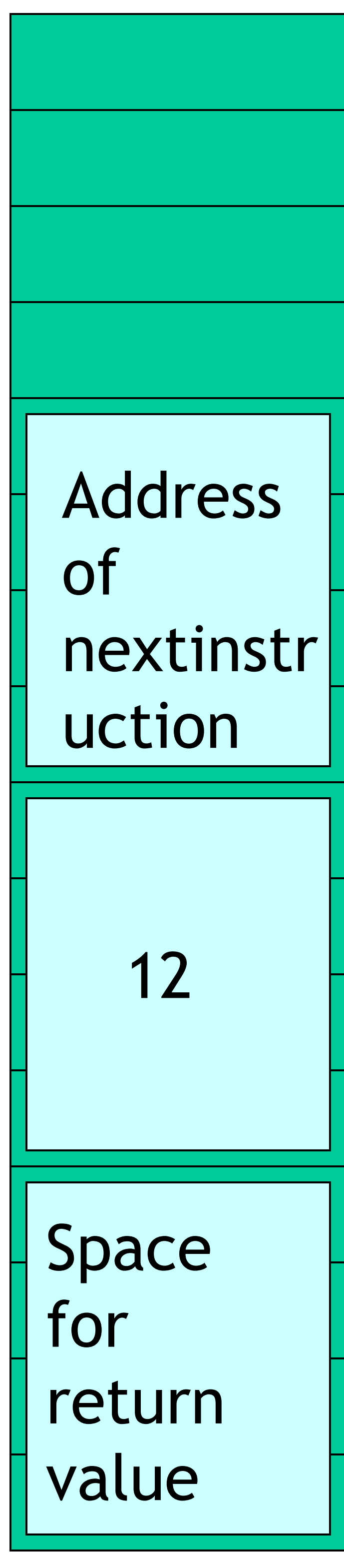
Copyright 2000, Douglas Reeves

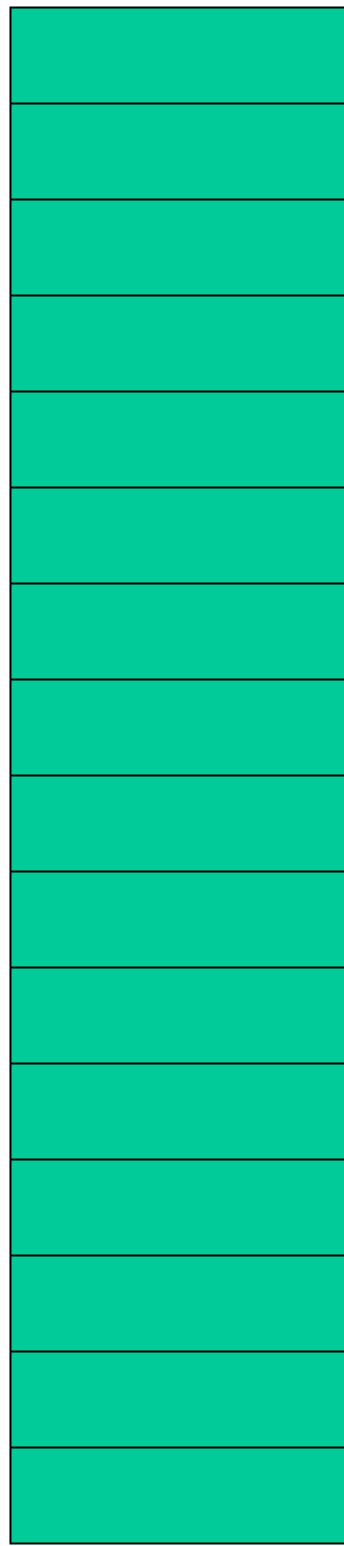
```
main:
        ….
        sub     ESP, 4 ; allocate 4 bytes
                       ; for the return value
        mov     maxch, 12
        push    maxch  ; maximum number
                       ; of characters to map
        call    upcase
nextinstruction:
        pop     nummap ; get return value + de-
                       ; allocate return val. space
        …
```


Copyright 2000, Douglas Reeves

Just after call to “upcase”

ESP →





The stack

Copyright 2000, Douglas Reeves

```
upcase proc
        push    EBP
        mov     EBP, ESP
        push    EAX         ; save EAX value
        push    ECX
        mov     ECX, 0
loop1:
        cmp     ECX, [EBP+8]
        je      endup
        get_ch
        cmp     AL, ‘a’
        jl      endup
        cmp     AL, ‘z’
        jg      endup
        add     AL, 23
        put_ch
        inc     ECX
        jmp     loop1
endup:
        mov     [EBP+12], ECX
        pop     ECX
        pop     EAX         ; restore EAX value
        pop     EBP         ; restore EBP value
        ret     4           ; deallocate input
                            ; parameter space
upcase endp
```

Just before "loop1"

EBP →

EBP+4 →

EBP+8 →

EBP+12 →

Saved "old" EBP value
Address of nextinstruction
12
Space for return value

ESP →

EBP-4 →

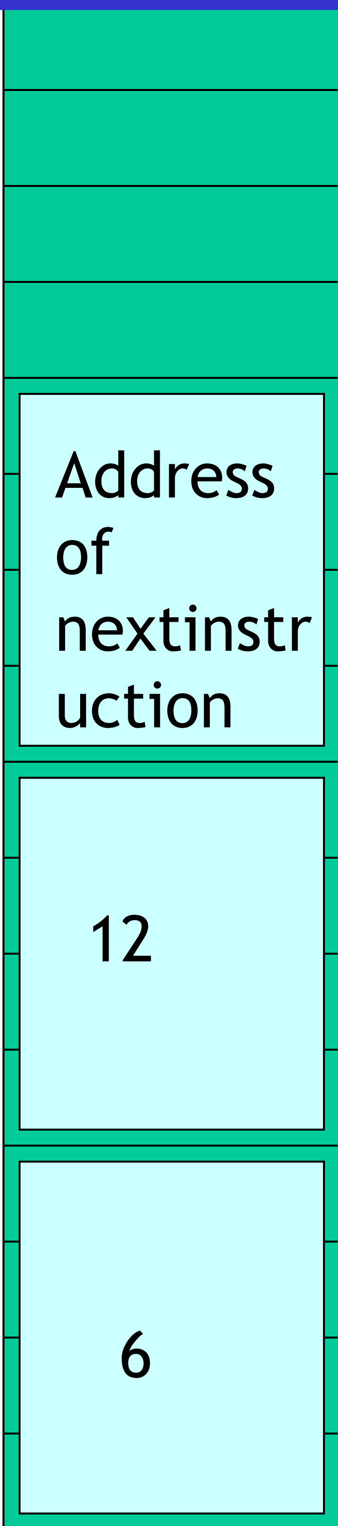
Saved "old" ECX value
Saved "old" EAX value

The stack

Copyright 2000, Douglas Reeves

Just before "ret 4"

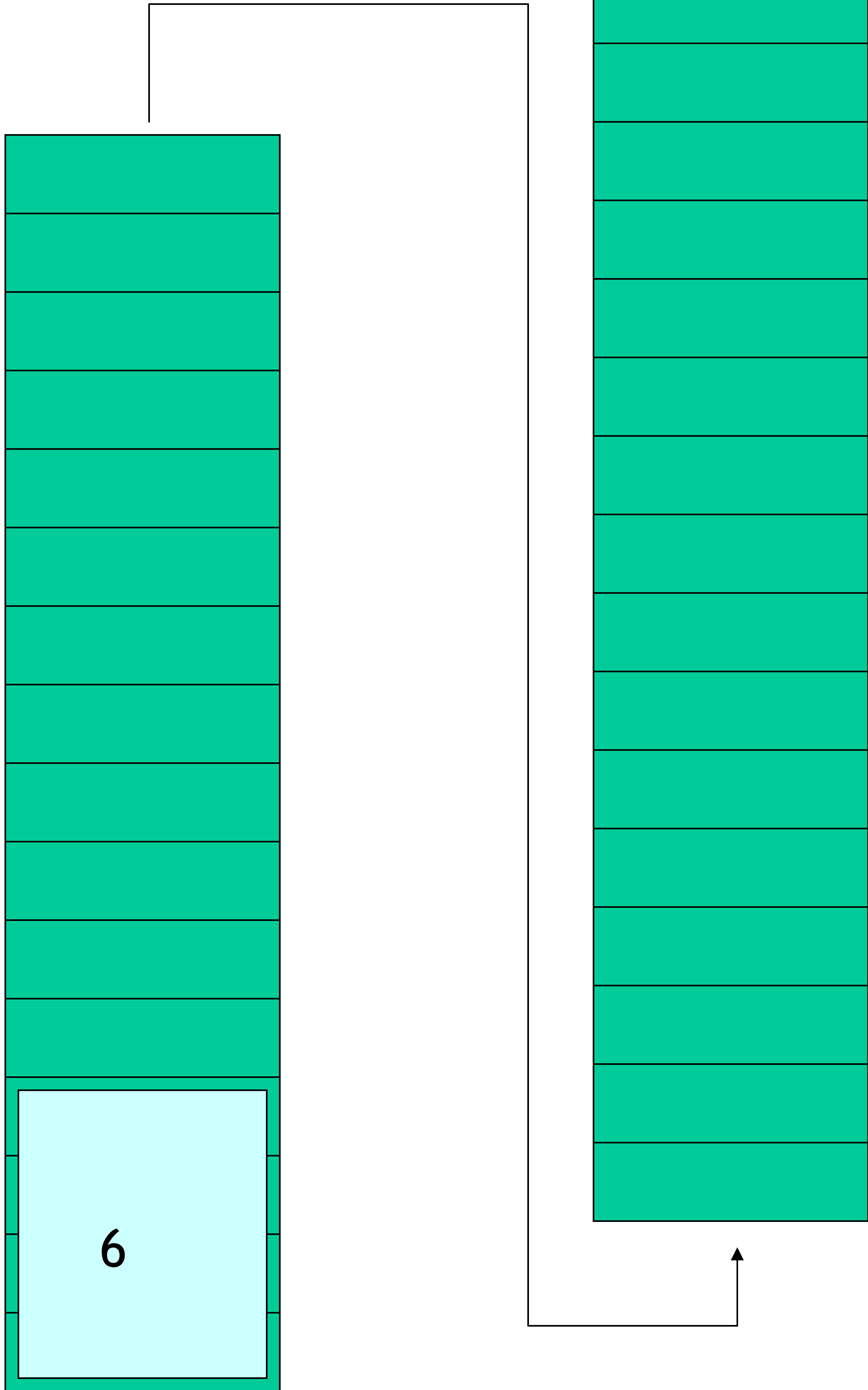
(assuming 6 characters mapped to upper case)



The stack

Copyright 2000, Douglas Reeves

Just after "ret 4"



Copyright 2000, Douglas Reeves

Procedure "Checklist"

1. Caller allocates space for return value
2. Caller copies input parameters to stack using push
3. "call" statement
4. Callee pushes EBP
5. Caller copies ESP to EBP
6. Caller saves registers it will modify
7. Caller allocates space for local variables
8. --Body of procedure executes--

Copyright 2000, Douglas Reeves

Programming Checklist (cont.)

9. Callee copies result into return variable space
10. Callee deallocates space for local variables
11. Callee restores registers, pops off stack
12. Callee restores EBP, pops off stack
13. Callee returns, deallocates input parameters from stack
14. Caller copies return value
15. Caller deallocates space for return value

Copyright 2000, Douglas Reeves

Using Registers, Instead

- How would the above example change if…
  - the procedure didn't call any other procedure (no nesting)?
  - + we used registers to hold input parameters?
  - + we used a register to hold the return value?
  - + the procedure used registers for all local variables?
- A lot simpler! If only we had enough registers...

Copyright 2000, Douglas Reeves

Example, Using Registers

```
main:
        ....
        mov     EDX, 12
        call    upcase
        mov     nummap, ECX
                        ; value of ECX was
                        ; changed by procedure
nextinstruction:
        ...
```


Copyright 2000, Douglas Reeves

```
upcase proc
        push    EAX         ; save EAX value
        mov     ECX, 0
loop1:
        cmp     ECX, EDX
        je      endup
        get_ch
        cmp     AL, 'a'
        jl      endup
        cmp     AL, 'z'
        jg      endup
        add     AL, 23
        put_ch
        inc     ECX
        jmp     loop1
endup:
        pop     EAX         ; restore EAX value
        ret
upcase endp
```


Copyright 2000, Douglas Reeves

Recursion

- Procedure calling itself!
- Traversing a tree, divide and conquer algorithms, etc.
- Example in C:

```
Int     f, a;
        a = 10;
        f = factorial(a);
…
Int     factorial(int a)
{
        int     b;
        if (a == 1)
                return(1);
        else {
                b = factorial(a-1);
                return (a * b);
        }
}
```

Example of Recursion

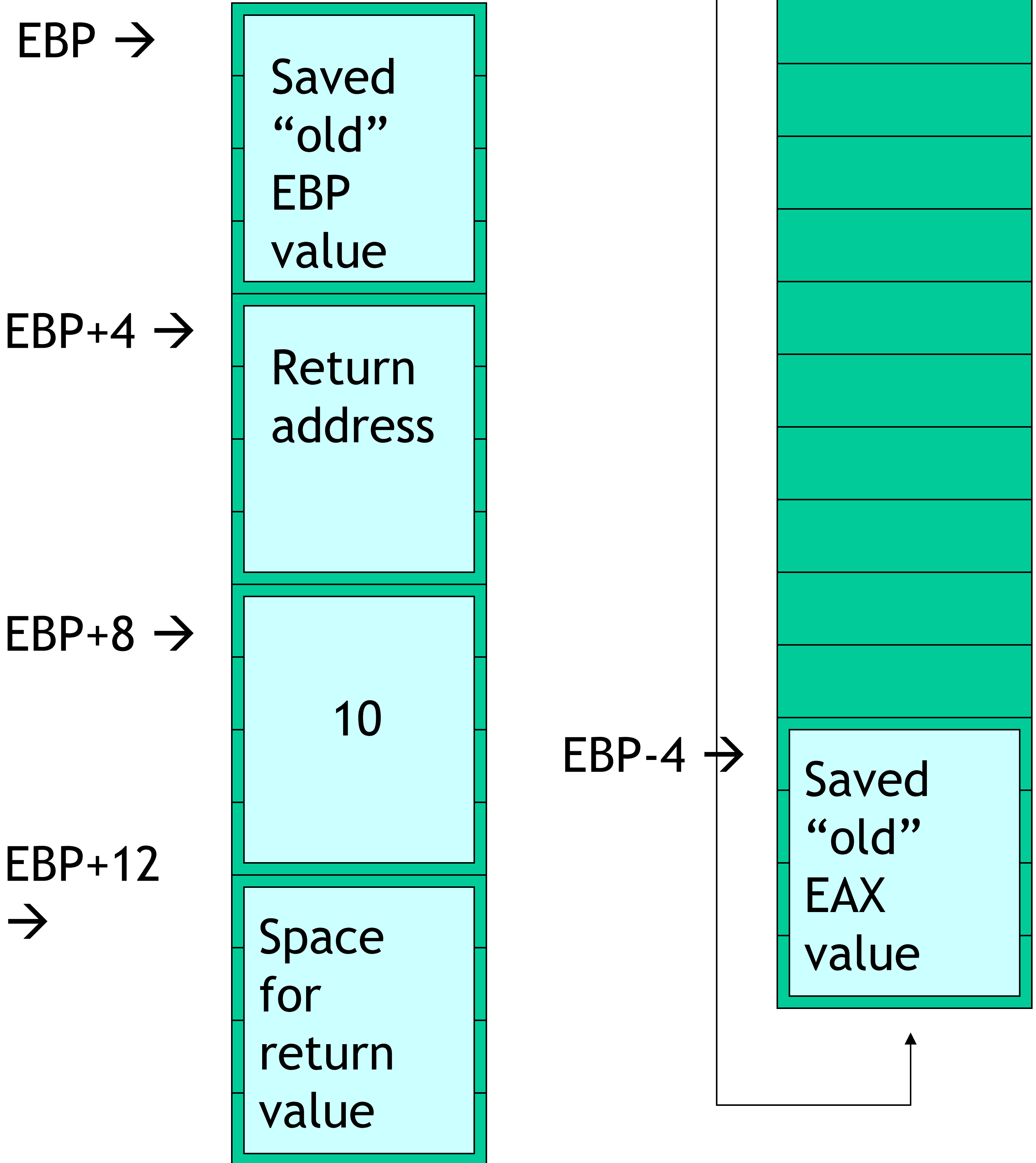
main:

```
        ....
        sub     ESP, 4 ; allocate 4 bytes
                       ; for the return value
        mov     a, 10
        push    a      ; push input parameter
                       ; on stack

        call    factorial

        pop     f      ; get return value off the
                       ; stack
        ...
```


Copyright 2000, Douglas Reeves

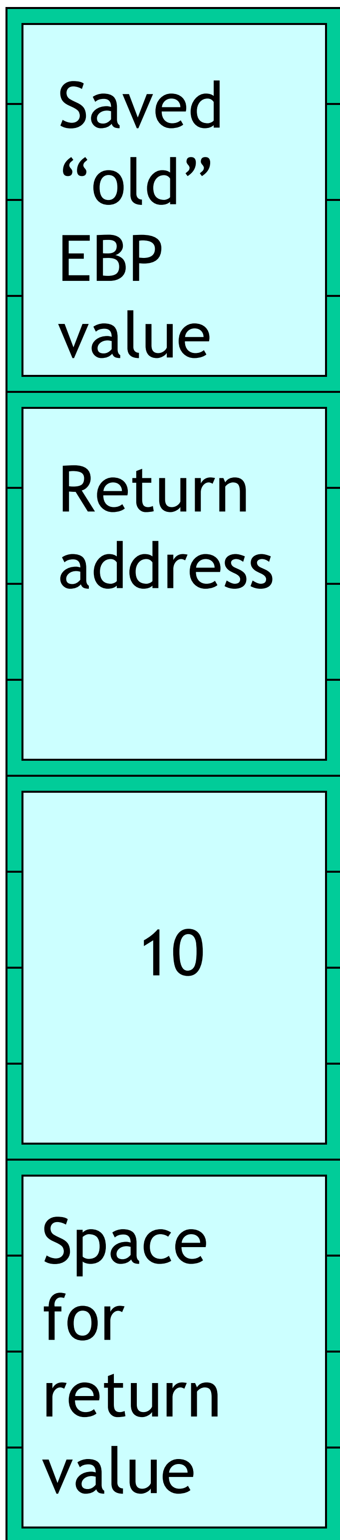
```
factorial proc
        push    EBP         ; save base pointer
        mov     EBP, ESP ; initialize base pointer
                            ; for this activation record
        push    EAX         ; save EAX value on stack

        cmp     [EBP+8], 1 ; check if input parameter = 1
        jne     elsepart  ; if not, keep going
        mov     [EBP+12], 1 ; otherwise, return value = 1
        jmp     returnall

elsepart:
        sub     esp, 4      ; allocate 4 bytes for return value
        mov     EAX, [EBP+8] ; decrement input parameter
        dec     EAX         ; and push it on the stack for
        push    EAX         ; the recursive call

        call    factorial ; here's the recursion

        pop     EAX         ; get the return value off stack
        mul     [EBP+8] ; multiply by input, a
        mov     [EBP+12], EAX ; store the result

returnall:
        pop     EAX         ; restore EAX value
        pop     EBP         ; restore EBP value
        ret     4           ; deallocate input
                            ; parameter space and return
factorial endp
```

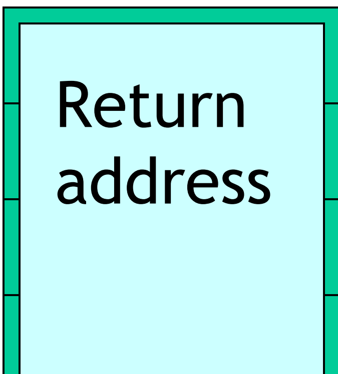
During first call to factorial(10)

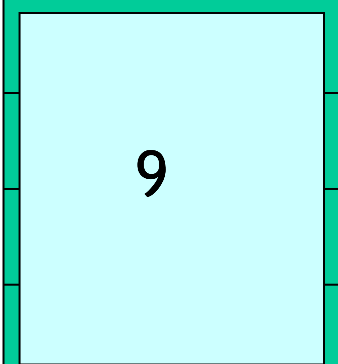


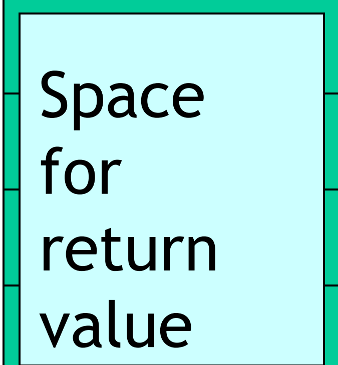
Copyright 2000, Douglas Reeves

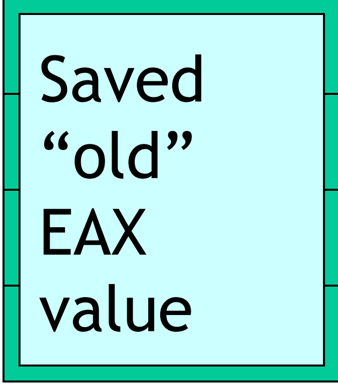
During recursive call to factorial(9)











EBP+4 →

EBP+8 →

EBP+12 →

The stack

Copyright 2000, Douglas Reeves