Procedure Calls

October 27

CSC201 Section 002

Fall, 2000

Advantages of Procedures

- Convert big, hard problem into smaller, easier sub-problems
- Easier to understand and debug
- Avoid redundancy and inconsistency problems
- Putting frequently-used procedures in libraries encourages reuse

Copyright 2000, Douglas Reeves

Procedures, Attempt #1

- Writing a procedure that is called from only one place in the program

```
main:
        ....
        jmp     myproc
nextinstruction:
        add     EAX, EDX
        ...

myproc:
        mov     EDX, myvar
        ...
        jmp     nextinstruction
```


Copyright 2000, Douglas Reeves

Procedures, Improved #1

- Writing a procedure that is called from multiple places in program

```
main:
        ....
        lea     EBX, nextinstruction
        jmp     myproc
nextinstruction:
        add     EAX, EDX
        ...
        lea     EBX, anotherinstruction
        jmp     myproc
anotherinstruction:
        sub     ECX, EDX


myproc:
        mov     EDX, myvar
        ...
        jmp     [EBX]
```

Procedures, Rule #1

- Requirement: save return address before jumping to procedure
  - Return address = address of *next* instruction after the jump to the called procedure
  - Jumping to the "return address" resumes execution in the calling routine

Copyright 2000, Douglas Reeves

Procedure Nesting

Procedure A calls

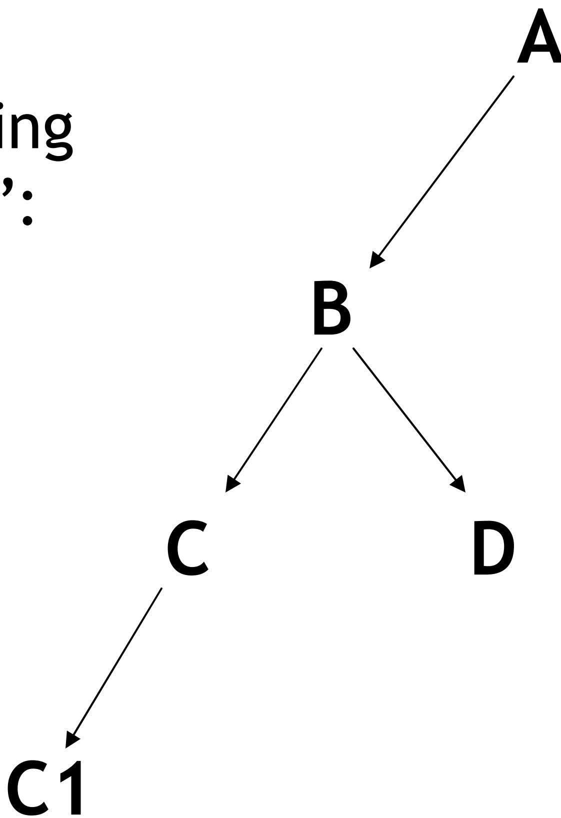
Procedure B which calls

Procedure C which calls

Procedure C1

Procedure D

A calling "tree":



- What's the problem with Version #1 if nesting is allowed?

Copyright 2000, Douglas Reeves

Using the Stack

- Each caller pushes return address on the stack
  - callee retrieves return address by popping it from the stack
- Advantage: return address is not overwritten when calls are nested!

Copyright 2000, Douglas Reeves

```
main:
        ....
        lea     EBX, nextinstruction
        push    EBX
        jmp     myproc
nextinstruction:
        add     EAX, EDX
        ...
        lea     EBX, anotherinstruction
        push    EBX
        jmp     myproc
anotherinstruction:
        sub     ECX, EDX

myproc:
        mov     EDX, myvar
        ...
        pop     EBX
        jmp     [EBX]
```


Copyright 2000, Douglas Reeves

A Quicker Way to Save and Restore Addresses

- "call procname" -- push return address (next instruction) on stack and jump to procname
- "ret" -- pop return address from stack and jump to that location

Copyright 2000, Douglas Reeves

```
main:
        ....
        call    myproc
; nextinstruction: label not needed!
        add     EAX, EDX
        ...
        call myproc
; anotherinstruction: label not needed!
        sub     ECX, EDX

myproc:
        mov     EDX, myvar
        ...
        ret
```


Copyright 2000, Douglas Reeves

Proc and Endp Directives

- Syntax:

```
mypproc     proc
        <body of procedure>
mypproc     endp
```

- Clearly delimit the start and end of procedure definitions

Copyright 2000, Douglas Reeves

Single Entry – Single Exit

- Warning #1: DON'T jump into the middle of procedures!
  - may not properly save registers, allocate local memory, etc.
- Warning #2: DON'T exit a procedure by jumping to an address outside the procedure!
  - may not pop the return address from the stack; stack corrupted!

Copyright 2000, Douglas Reeves

Procedure Comments (Recommended)

- Purpose
- (Author + Date)
- Parameters used
- Return value
- Registers affected

Copyright 2000, Douglas Reeves

Local Variables

- Variables are defined locally to a procedure because global variables are bad!
  - The "scope" of the variable is within just that procedure
  - Dynamically created on entry to procedure, deallocated on exit

Copyright 2000, Douglas Reeves

Example (C language):

```
char c;
int     i1, j1, k1;

main() {
        c = getc();
        i1 = myproc();
        …
}

int myproc() {
        char   c;
        int    m;
        c = getc();
        m = (int) c – 25;
        return(m);
}
```


Copyright 2000, Douglas Reeves

Local Variables

- Registers are also global variables; not a good way to pass values!
- Statically allocated memory is global; also not good!
- Door #3: use the stack for dynamic memory allocation for local variables

Copyright 2000, Douglas Reeves

Activation Records

- "Activation record" = all the stuff put on the stack during a procedure call
  - Return address
  - Any saved data (note: more later)
  - Local variables
- Activation records are popped on procedure exit

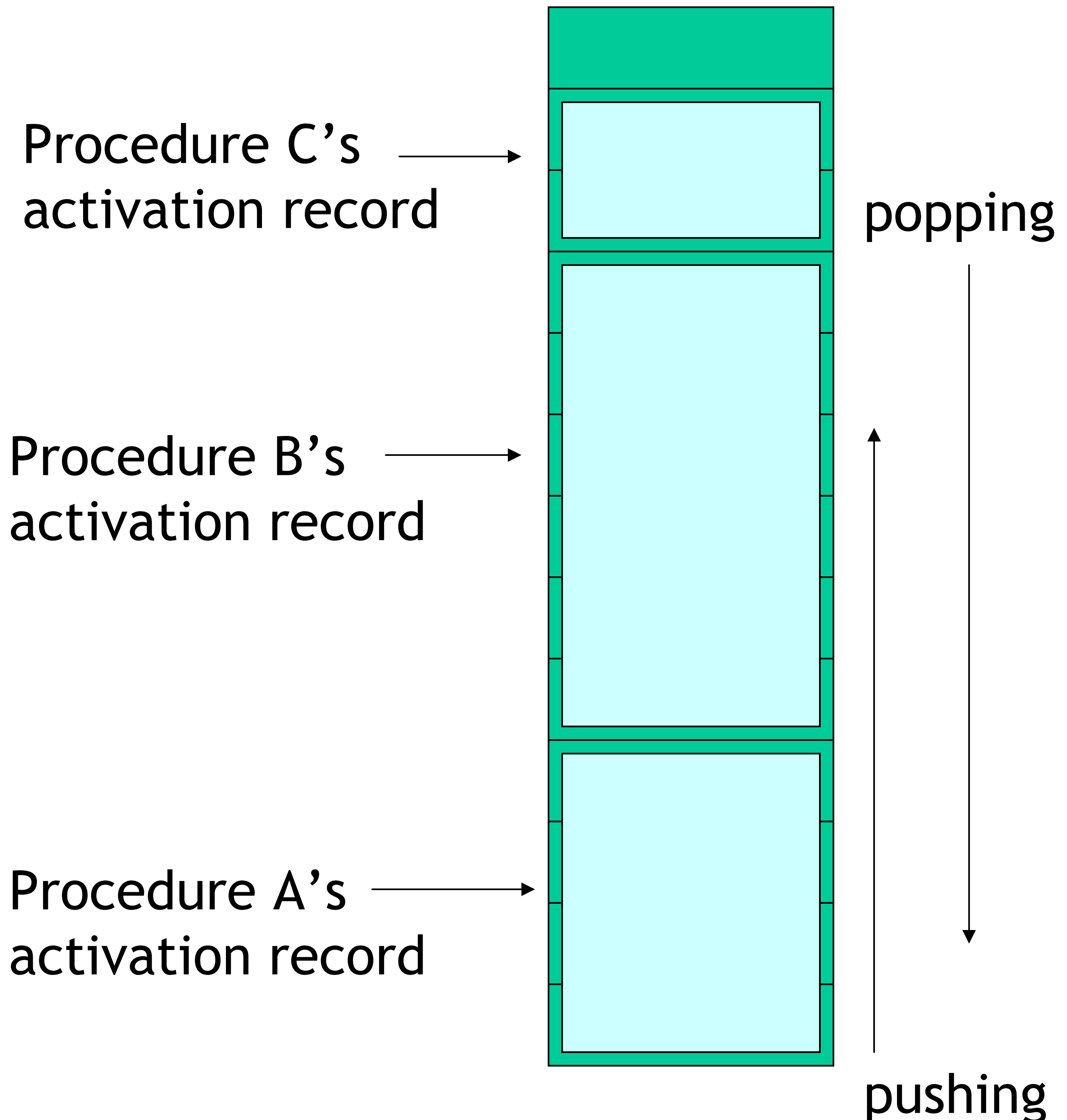
Copyright 2000, Douglas Reeves

Example

Procedure A has called

Procedure B which has called

Procedure C



Copyright 2000, Douglas Reeves

Space Allocation and Addressing

- Allocation = decrement the stack pointer (ESP)
  - Size of the decrement = amount of memory allocated for temporary storage

```
dec     ESP, 20
```

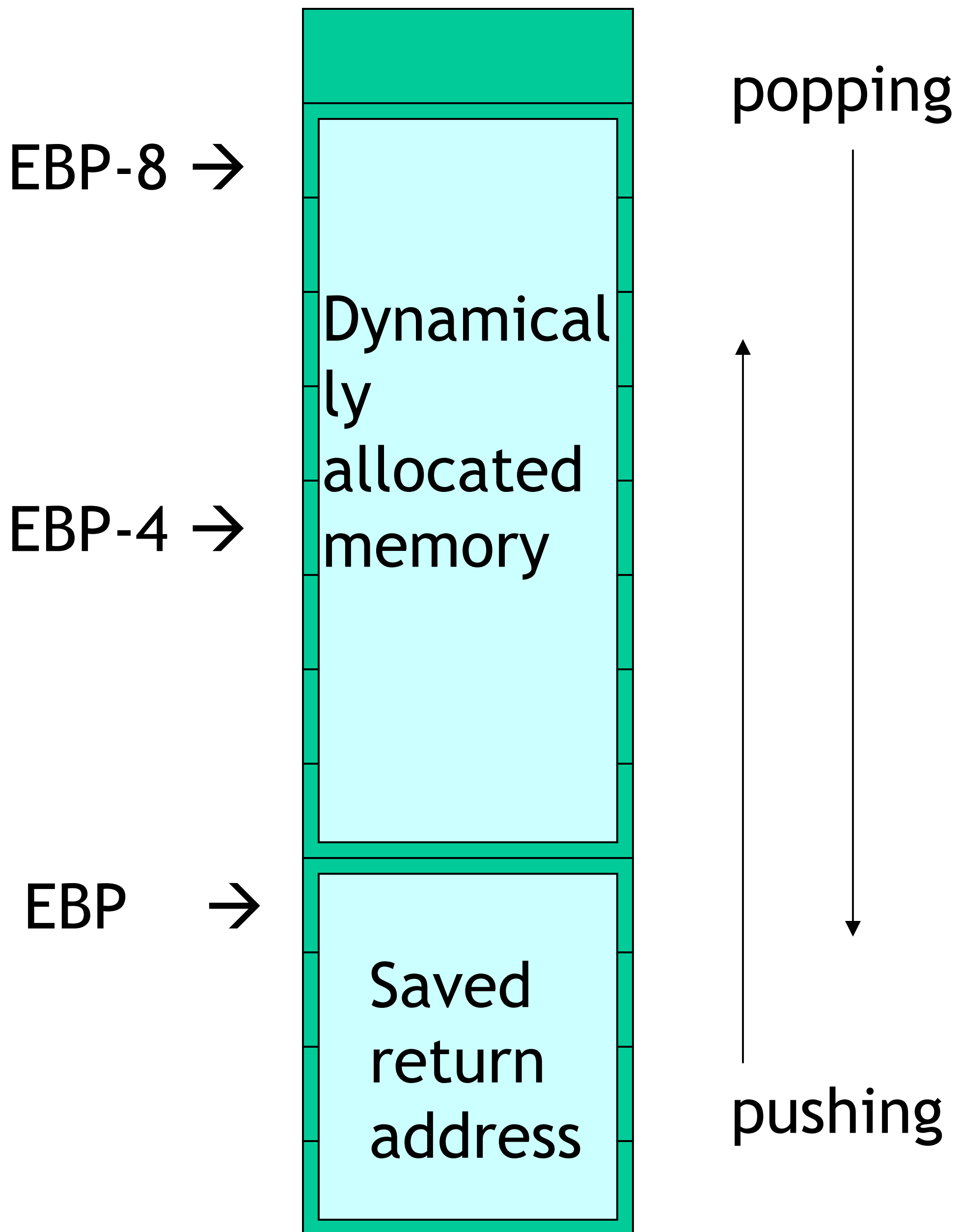
- How refer to (address) a word of memory in this area?
  - First, copy ESP value into EBP
  - Then, use based-displacement addressing
  - EBP = base, displacement always negative (why?)

Copyright 2000, Douglas Reeves

Stack Addressing

```
Myproc      proc
      mov   EBP, ESP
      dec   ESP, 20
      ...
      mov   dword ptr [EBP], 1000
      mov   dword ptr [EBP-4], 25
      ...
```


Copyright 2000, Douglas Reeves



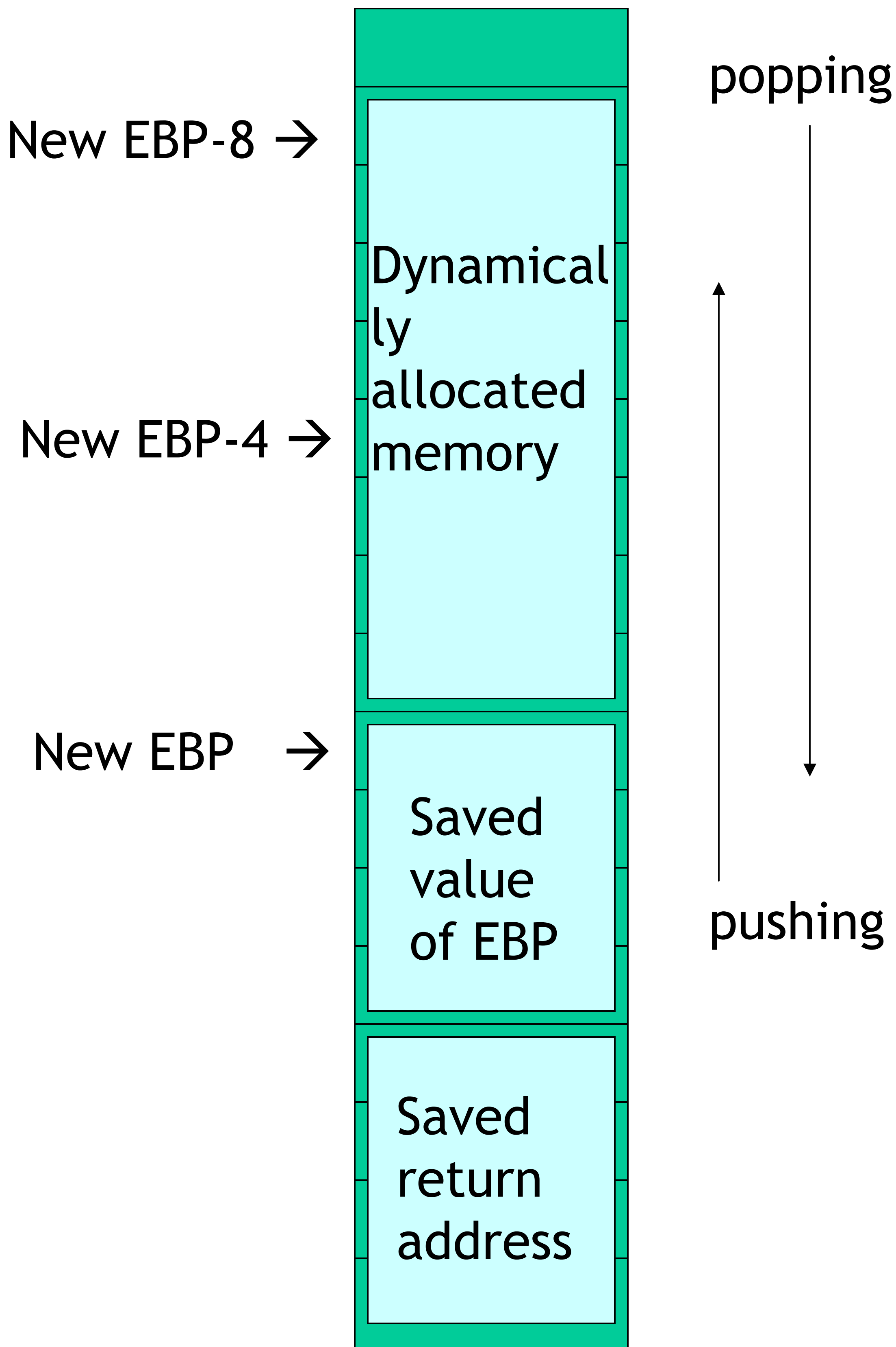
The stack

Copyright 2000, Douglas Reeves

However...

- Don't forget to save the "old" EBP value first
  - Push EBP on stack

Copyright 2000, Douglas Reeves



Copyright 2000, Douglas Reeves