Subprograms I

Ned Nedialkov

McMaster University
Canada

SE 3F03
February 2014

Outline

Passing parameters

Parameters can be passed through

- registers
- stack
- registers + stack

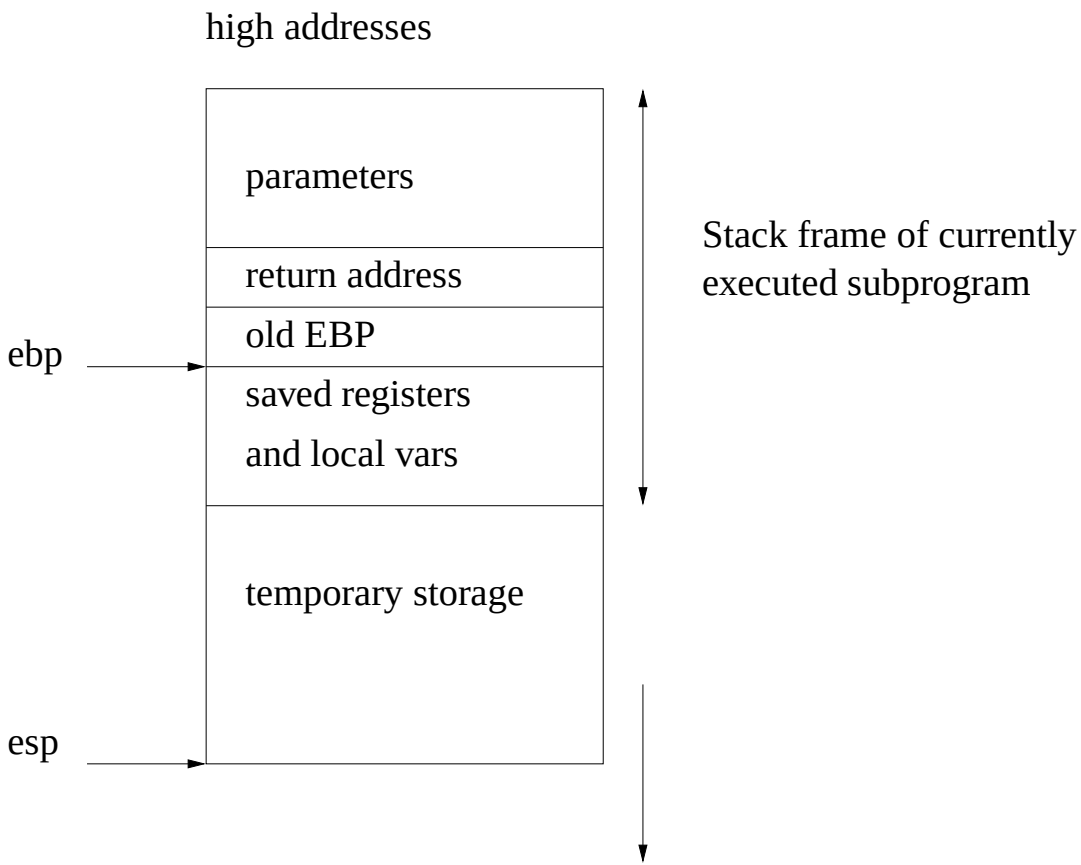
Local variables: where to store them?

Stack frame

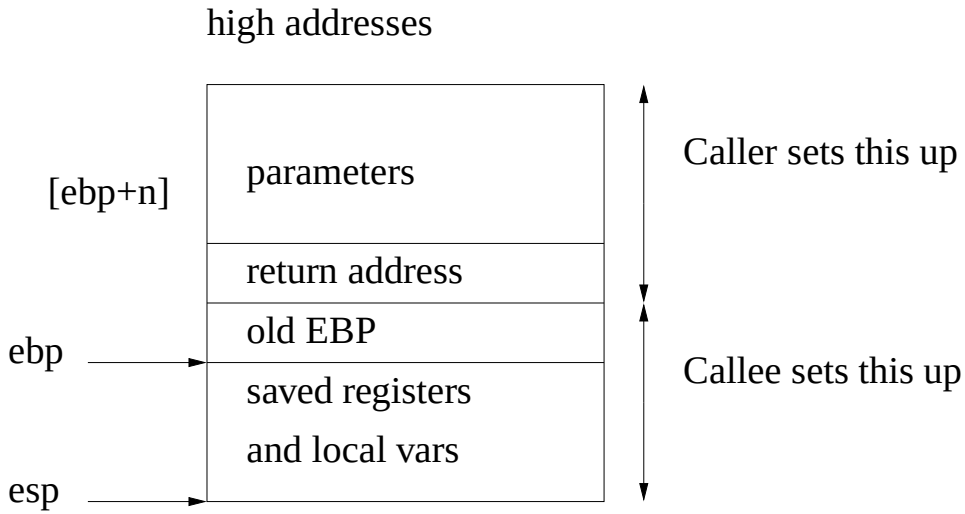
The stack frame is a fixed block of memory for storing

- parameters
- return addresses
- local variables

- **ebp** base pointer: points to a fixed position in the stack frame
- **esp** stack pointer: points to the top of the stack



The stack frame grows towards lower addresses


high addresses
parameters
[ebp+n]
return address
old EBP
ebp
saved registers
and local vars
esp
Caller sets this up
Callee sets this up

Push/Pop

- **push eax**
  - **esp** <-- **esp**-4
  - stores **dword**, here **eax**, at **esp**
- **pop eax**
  - reads a **dword** at **esp** and stores it into **eax**
    this word is not physically removed
  - **esp** <-- **esp**+4
- Example

```
;; assume esp = 1000h
push dword 1              ;esp=0FFCh
push dword 2              ;esp=0FF8h
push dword 3              ;esp=0FF4h
pop eax                   ;eax=3, esp=0FF8h
pop eax                   ;eax=2, esp=0FFCh
pop eax                   ;eax=1, esp=1000h1
```

- **pusha** stores in the stack **eax**, **ebx**, **ecx**, **edx**, **esi**, **edi**, **ebp**
- **popa** restores them
- Ensure data is pushed and restored correctly!

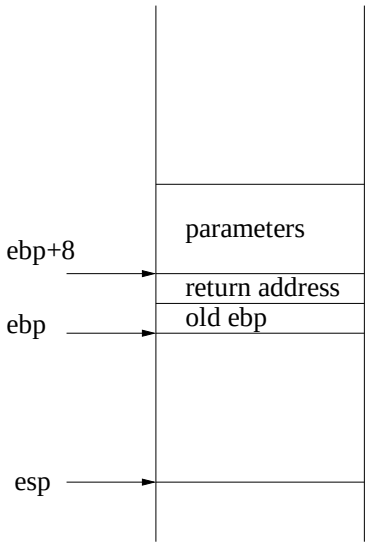
Call/Return

- **call label**
  - pushes the address of the next instruction to be executed into the stack
  - transfers program execution to **label** (unconditional jump)
- **ret**
  - pops of the top of the stack
  - transfer control to the address stored at the top of the stack (unconditional jump)

Calling conventions

- Caller: pushes parameters onto the stack
- Callee: accesses them in the stack
- **ebp** is saved by the callee
  - **push ebp**
  - **mov ebp, esp**
  - **ebp** is fixed, **esp** can change
- Structure

```
subroutine:
        push ebp
        mov ebp, esp
        ;;
        ;; instructions
        ;;
        pop ebp
        ret
```



- Parameters always start at **ebp**+8
- Parameters are at **ebp**+n
- Local variables are at **ebp**-n
- After a subprogram is done, parameters must be removed from the stack. Who does this?

C calling conventions

- The caller puts the parameters into the stack and removes them
- Reason: varying number of parameters
- Example

```
push dword 1
call fun
add esp, 4
```

  - No need to do **pop**; the compiler may do **pop ecx**, shorter instruction
- Local variables are located after **ebp**

```
subroutine:
        push ebp
        mov ebp, esp
        ;; move the stack pointer down to
        ;; allocate space for local variables
        sub esp, n
        ;;
        ;; instructions
        ;;
        pop ebp
        ret
```

- n must be a multiple of 4
- Disadvantage: if a subprogram is called many times, the caller must clear the space for parameters each time
- Pascal: fixed number of parameters. The callee clears this space

Enter/Leave

- **enter** `a,b`
  - creates a stack frame
  - a amount of stack space
  - b nesting level
- **enter** `a,0` is the same as

```
push ebp
mov ebp,esp
sub esp,a
```

- **leave**
  - clears a stack frame
  - same as

```
mov esp,ebp
pop ebp
```