Subprograms - Part 1

CS 2XA3

Term I, 2018/19

Outline

Passing parameters

Parameters can be passed via

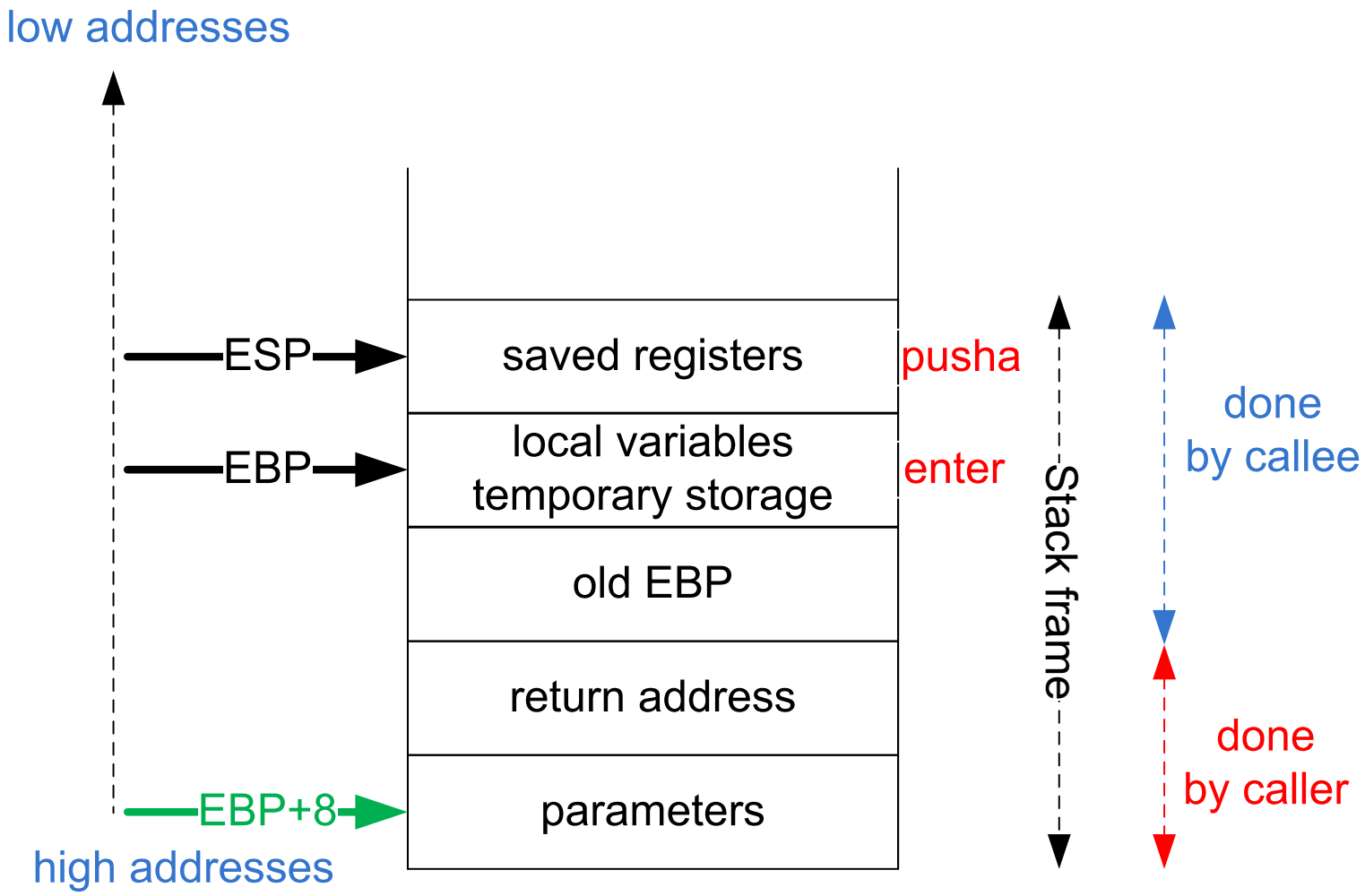
- registers
- stack (control stack, system stack)
- registers+stack

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
- **`esp`** *stack pointer*: points to the top of the stack frame



The stack frame grows towards lower addresses

Push/Pop

- **push eax**
  - **esp** ← **esp-4**
  - stores **dword** (here **eax**) at **esp**
- **pop eax**
  - reads a **dword** at **esp** and stores it in **eax**
  - the 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=1000h
```

- **`pusha`** stores on the stack **`eax`**, **`ebx`**, **`ecx`**, **`edx`**, **`esi`**, **`edi`**, **`ebp`**
- *mnemonics*: **`pusha`**=push all
- **`popa`** restores them
- *mnemonics*: **`popa`**=pop all
- They ensure that 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
  - transfers control to the address stored at the top of the stack (unconditional jump)

Calling conventions

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

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

C calling conventions

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

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

  No need to do `pop`*; a compiler may do* `pop ecx`*, but* `add` *is a 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

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
  - the same as

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