5. Procedures

Tuesday, September 10, 2024 11:36 AM

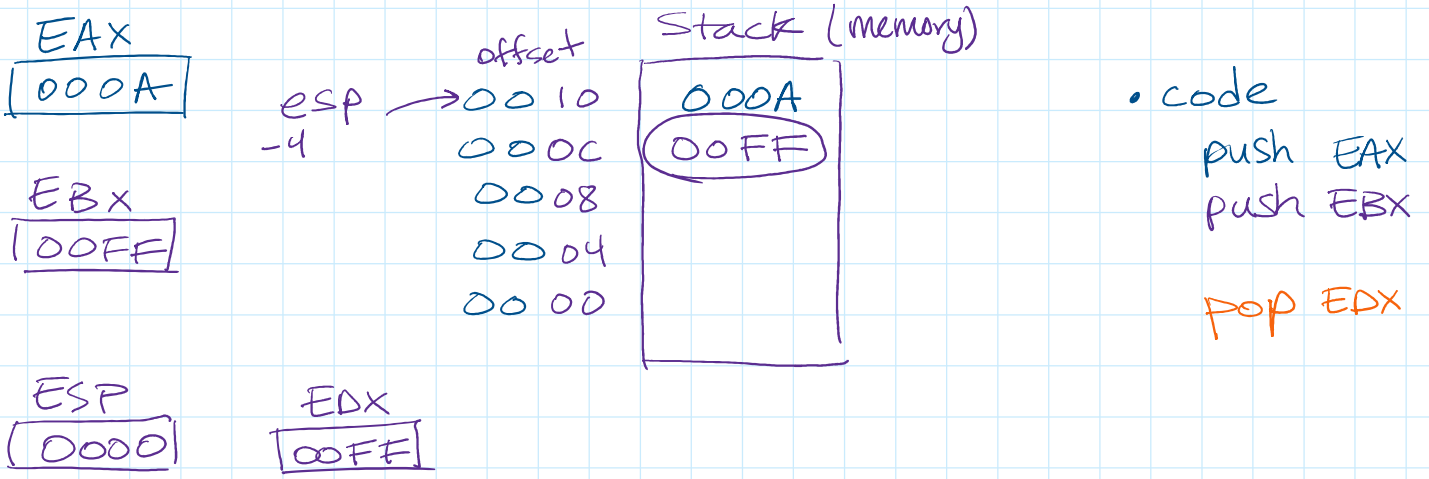
* A procedure is a block of statements separate from other procedures.
  * subroutine, function, method

* Stack
  * stack data structure - LIFO (last in, first out)

* Runtime stack - memory array managed by CPU.
  * Has preallocated amount of memory
    * stack 4096 - allocates 4096 bytes to runtime stack

* Keeps track of:
  * subroutine addresses
  * procedure parameters
  * local variables

* ESP is a 32-bit register (stack pointer)
  * Don't manipulate ESP directly
  * Is indirectly modified by the CALL, RET, PUSH & POP instructions.
  * ESP always points (contains address) of last value that was added to stack.



* code

```
push EAX
push EBX

pop EDX
```

PUSH Operation

* decrements stack pointer (ESP) by amount according to operand's size & copies value into location in stack referenced by stack pointer

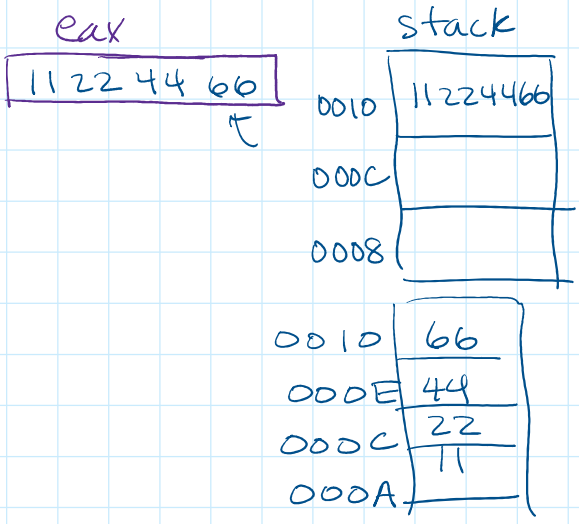
POP operation

* returns a copy of last value on stack and increments stack pointer (ESP) by amount according to instruction operand.

- code

```
push eax   → 4 bytes
push bx    → 2 bytes

pop bx
pop eax
```



PUSH reg/mem16
reg/mem32
imm32

POP reg/mem16
reg/mem32

* PROC Directive

* named block of statements that ends in a return statement
* PROC & ENDP Directives to start and end a procedure

example:

```
count PROC
  .
  .
  .
  RET
count ENDP
```

* Register parameters:
  * arguments to the procedure will be placed in a register before calling the procedure
  * returned values will also be placed in registers
  * Not a good idea to reference variables (memory) within procedures.

CALL Instruction

* call instruction pushes its return address onto the stack and copies the called procedure's address into instruction pointer (EIP)

RET Instruction

* ret instruction pops return address from stack into the instruction pointer (EIP)