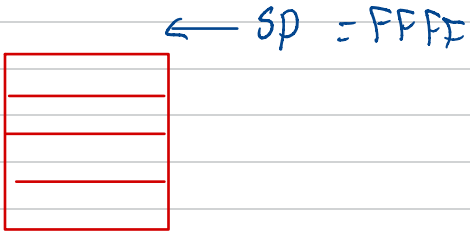
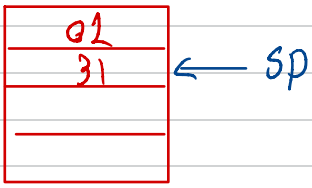
SP at start points to FFFF

nothing is stored at sp: FFFF



push Ax



Procedures are only Called using Call

Procedure	Macro
Must be invoked using "CALL " statement	invoked by just typing its name Does not need "CALL" statement
Fetching it takes more time due to the CALL and return	Fetching it takes less time
Executed once, and stored once in the memory	Executed each time it is needed, and duplicated in the memory
Time consuming	Memory consuming
Suitable for more than 10 instructions	Suitable for less than 10 instructions

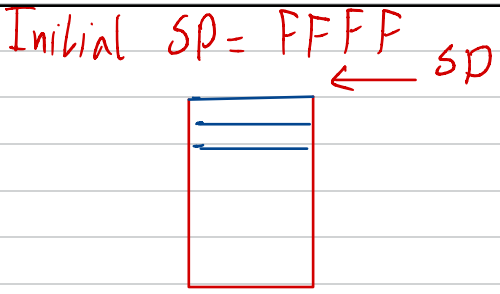
LOOP

- A combination of a **decrement CX** and the **JNZ conditional jump**.

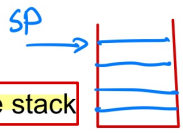
CONDITIONAL LOOPS

- LOOP instruction also has conditional forms: **LOOPE** and **LOOPNE**
- **LOOPE** (loop while equal) instruction jumps if CX != 0 while an equal condition (Z = 1) exists.
  - will exit loop if the condition is not equal or the CX register decrements to 0
- **LOOPNE** (loop while not equal) jumps if CX != 0 while a not-equal condition (Z = 0) exists.
  - will exit loop if the condition is equal or the CX register decrements to 0

STACK IMPLEMENTATION IN MEMORY (CONT.)



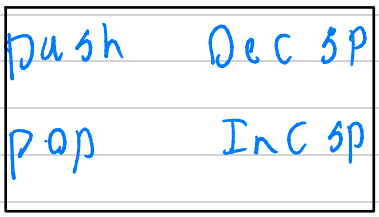
SS - Stack Segment

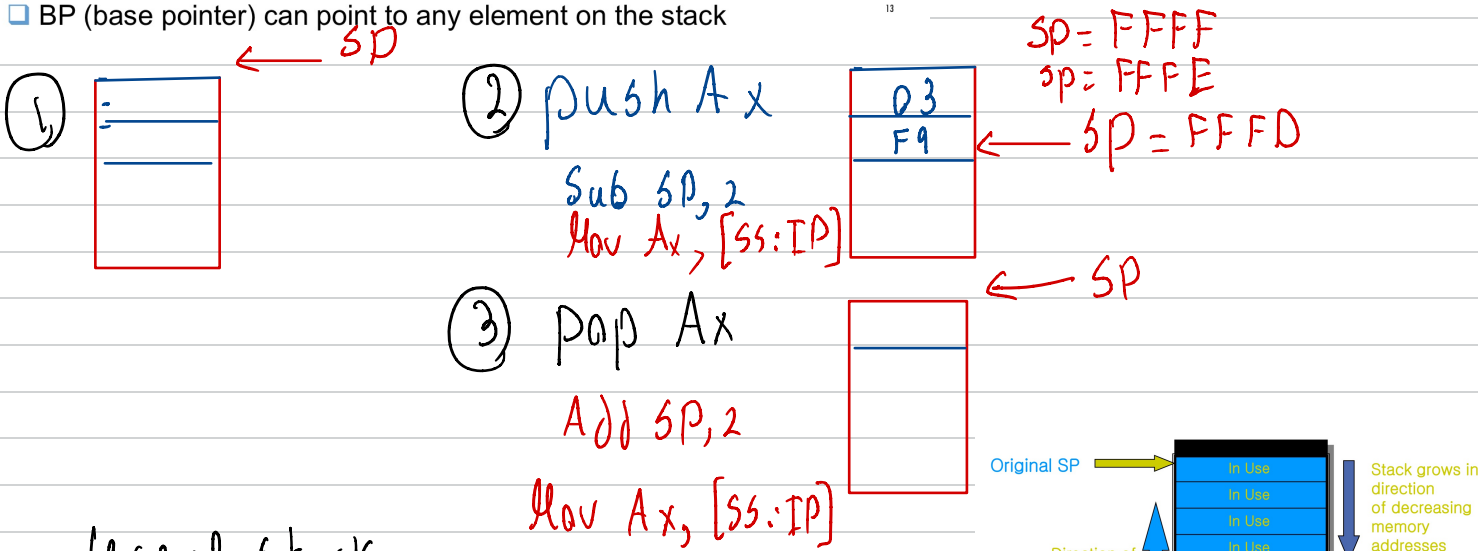


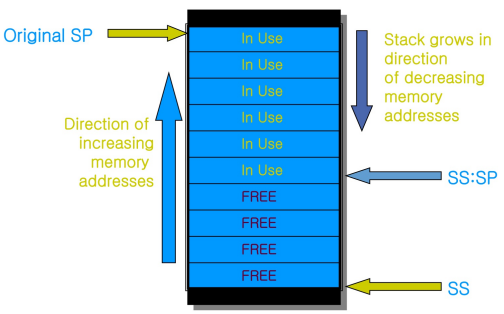
- SP (stack pointer) always points to the top of the stack
  - SP initially points to top of the stack (high memory address).
  - SP decreases as data is PUSHed

    PUSH AX ==> SUB SP, 2 ; MOV [SS:SP], AX
  - SP increases as data is POPed

    POP AX ==> MOV AX, [SS:SP] ; ADD SP, 2
  - BP (base pointer) can point to any element on the stack







Use of stack

- **To store**
  - registers
  - return address information while procedures are executing
  - local variables that procedures may require

- One point of access - the top of the stack
- A stack is always operated as Last-In-First-Out (LIFO) storage, i.e., data are retrieved in the reverse order to which they were stored.
- Instructions that directly manipulate the stack
  - **PUSH** - place element on top of stack
  - **POP** - remove element from top of stack

The top of the stack is where SP points to

---

* Procedures:

- CALL pushes the address of the instruction following the CALL (**return address**) on the stack.
  - the stack stores the return address when a procedure is called during a program

We use Call to use a procedure:

Call → 1) pushes the address of the next instruction (Instruction after Call) in the stack segment

2) It transfers Control to the Proc by updating the instruction pointer (IP) to the Address of the instruction

Related to where proc is stored or type of Call

Ret → 1) pops the return address [The one that was pushed] and moves it into IP

STORE RETURN ADDRESS OF A PROCEDURE

```
PrintRec   PROC   NEAR        → in CS
           ...
           <Print value of a record>
           ...
           RET
PrintRec   ENDP

main       PROC   FAR         → outside CS
           ...
           <Calculate Scores>
           ...
           CALL   PrintRec
           <Continue Execution HERE>
           ...
           CALL   DOSXIT
main       ENDP
```

At execution time

1. processor encounters the CALL to the procedure
2. pushes the return address (instruction pointe of the next instruction after the CALL) onto the stack
3. jumps to PrintRec
4. executes the code therein
5. pops the
6. returns to the calling coreturn address off th

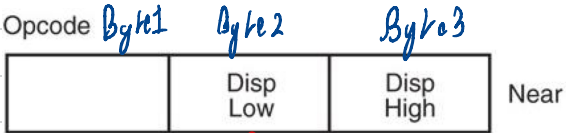
① Before calling the next address instruction saved inside stack segment

② After reaching the end of procedure

③ the saved Address is popped

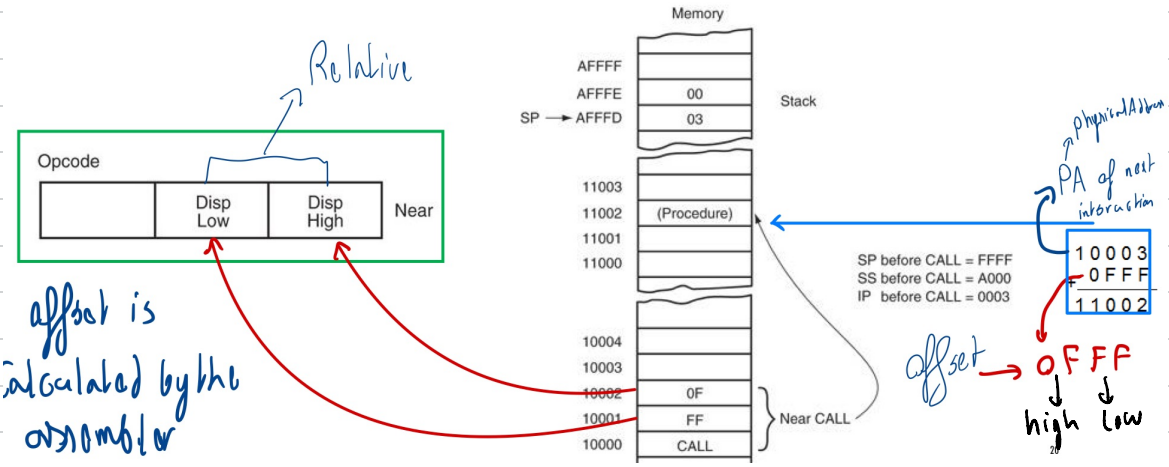
Types of Call

* Near Call: 3 bytes



NEAR CALL

The effect of a near CALL on the stack and the instruction pointer.

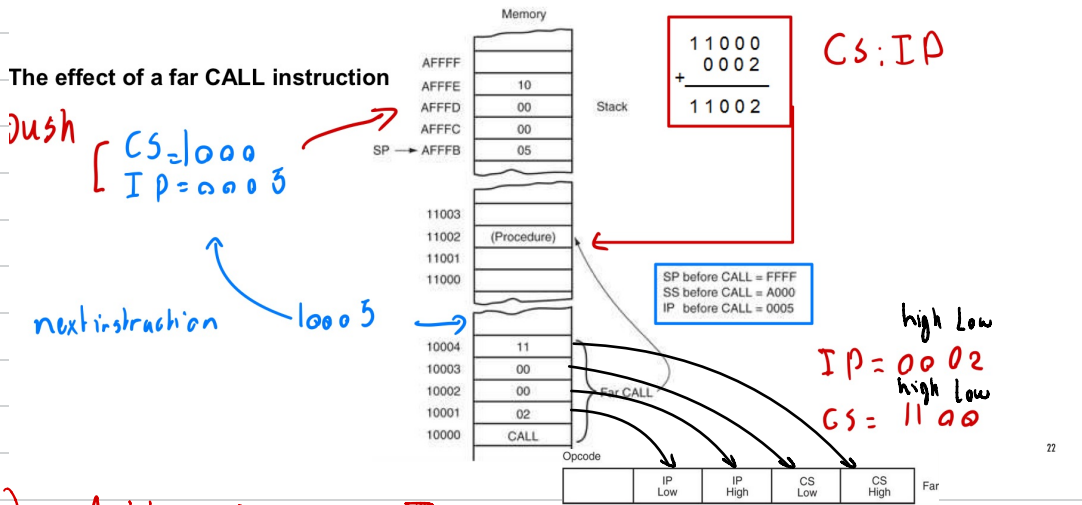


1) Address of next Instruction is pushed into the stack
   SUB SP, 2

2) The offset is added to PA [of Next instruction]
   The offset shows the relative distance from procedure to IP
   [It's Calculated by the assembler]

FAR CALL

The effect of a far CALL instruction



push [CS=1000, IP=0003]

next instruction → 1000 3

1) Address of next Instruction is pushed into the stack
   SUB SP, 2
2) Address of procedure CS:IP

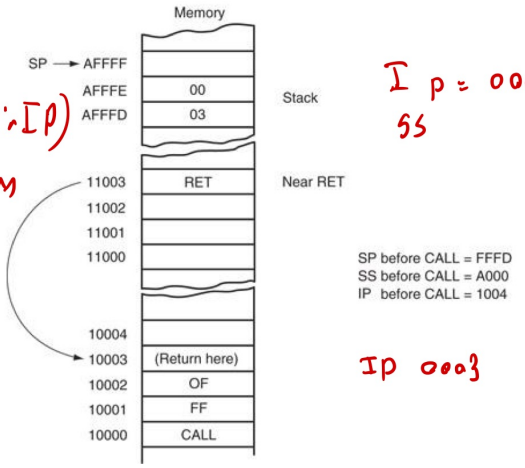
After Ret

RET

At the end of procedure

for Near Call

The effect of a near return instruction on the stack and instruction pointer



Pop IP (place first element in IP)

CS: IP → return to this Address

Sub SP, 2

Ip = 0003

SS

Ip 0003

---

for far Call

IP, CS are popped → stored in IP, CS respectively

Sub SP, 4

MACROS

function [Takes a Copy from these lines] without needing to store the next instruction is SS

- A macro inserts a block of statements at various points in a program during assembly

LOCAL VARIABLE(S) IN A MACRO →

Marco
Local Continue
Local cont 1

- A local variable is one that appears in the macro, but is not available outside the macro
- We use the LOCAL directive for defining a local variable

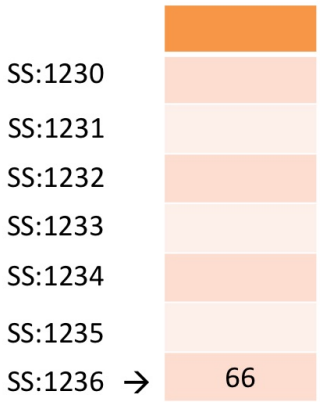
MACROS VS PROCEDURES (CONT.)

- **Advantage of using macros** for small number of line
  - execution of MACRO expansion is usually faster (no call and ret) than the execution of the same code implemented with procedures
- **Disadvantage**
  - assembler copies the macro code into the program at each macro invocation
  - if the number of macro invocations within the program is large then the program will be much larger than when using procedures

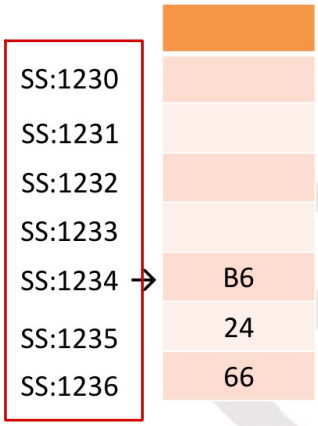
- SP points to the current memory location (top of the stack/ the latest thing you pushed).

- Example: Pushing onto the stack.

Assuming that SP = 1236, AX = 24B6. show the contents of the stack after the execution of PUSH AX

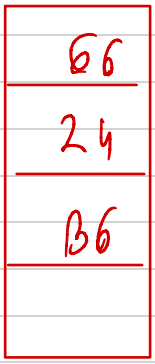


Start, SP = 1236



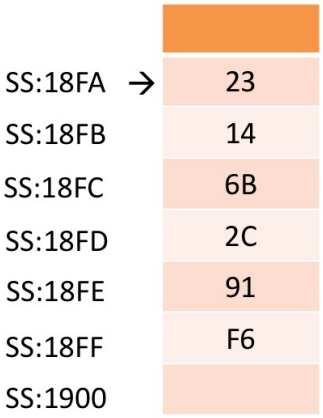
PUSH AX, SP = 1234

similar to

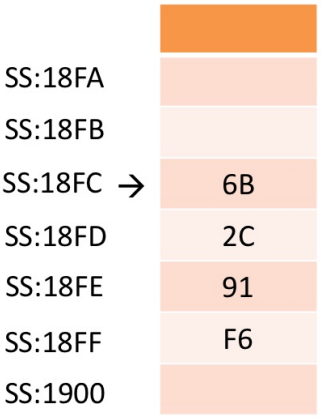


- Example: Popping the stack.

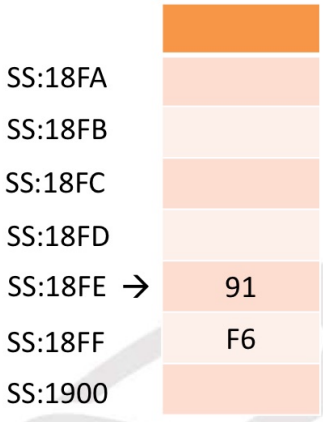
Assuming that SP = 18FA, AX = 24B6. show the contents of the stack after the execution of POP DX



Start, SP = 18fA



POP CX
SP = 18FC



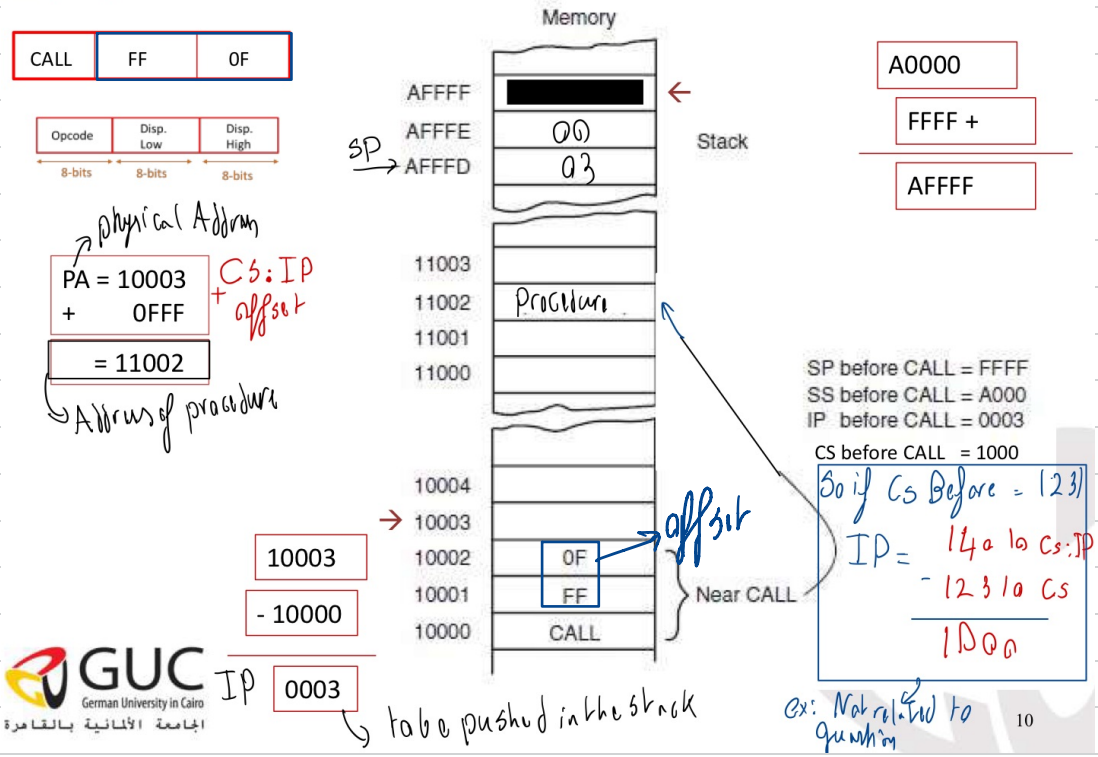
POP DX
SP = 18FE

1) Get offset from stack [near for Cs:IP]

2) push IP [of next instruction] into stack by [Cs:IP (PA) − CS (shifted) / IP]

2) Add offset to PA of next instruction after Call

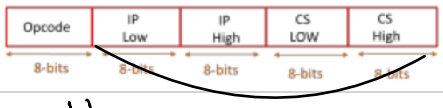
To get Address of the procedure



far call

1) Compute where sp points / PA of the stack

2) push CS, IP of next instruction into the stack

3) using 

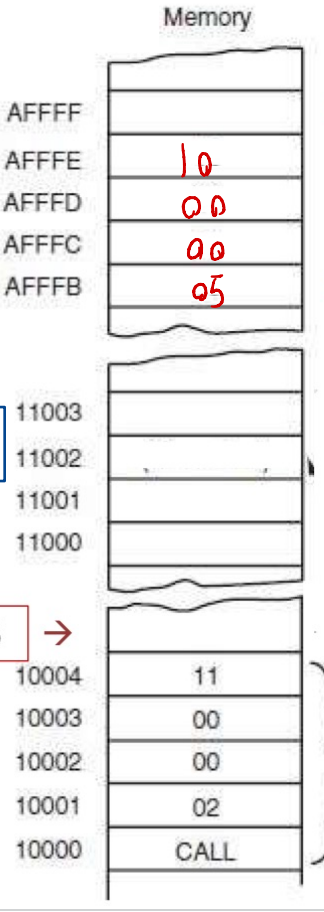
to get procedure Adress

PA = 11000 + 0002 = 11002 → Address of procedure

Ex: WHAT happens when FAR Call is executed?

First Pushing the CS and IP → of next instruction in the stack

Call | 02 | 00 | 00 | 11



initial ↑ SP

↑ PA of stack

A0000

FFFF +

AFFFF

← SP after push

next instruction ← 10005 →

10005

- 10000

0005

SP before CALL = FFFF
SS before CALL = A000
IP before CALL = 0005
CS before CALL = 1000

GUC German University in Cairo الجامعة الألمانية بالقاهرة