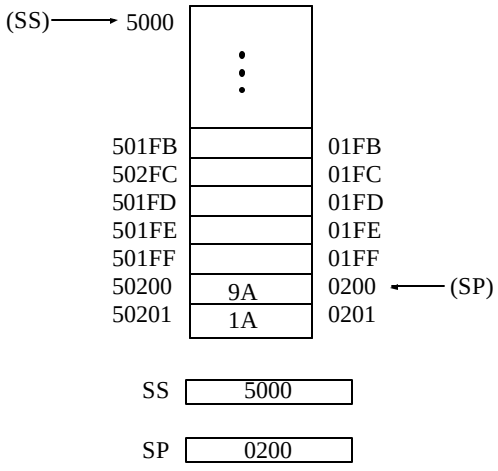
# Lecture 15: The Stack

- What is it?
- What is it used for?

# The Stack

- A special memory buffer (outside the CPU) used as a temporary holding area for addresses and data

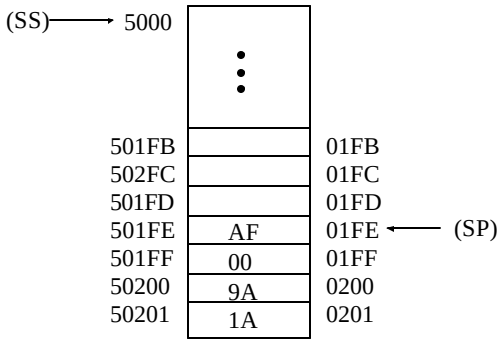
# The Stack




# Adding Elements

- Elements are added by *pushing* them on to the stack.

```
mov ax, 0AFh
push ax
```



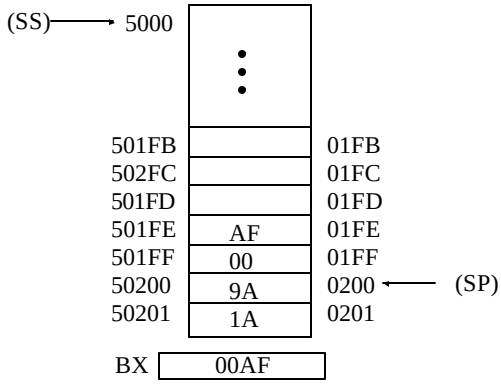
## Adding elements, cont.

- When data is pushed,

SP is left pointing to the item just pushed (the "top" of the stack)

## Removing (Retrieving) Information

- Elements are removed by *popping* them from the stack.
  - pop bx




## Retrieving, cont.

- When data is popped:

- SP is pointing to the new top of stack.

## A Word of Warning!

- The book draws the stack going from high memory to low memory:

  - picture from Irvine

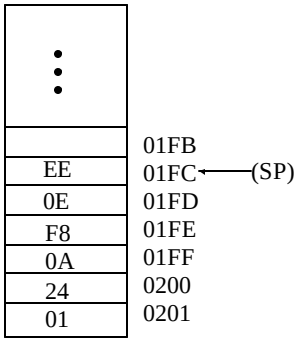
# Push

- PUSH decrements SP and copies a 16 bit or 32 bit register or memory operand onto the stack at the location pointed to by SP.
- Allowed forms:
  - push reg
  - push memval
  - push immed
- Examples:
  - push AX
  - push count ;where count dw ?
  - push 0a23h
- You can't push a byte on to the stack!

# Push Examples

```
;assume SP =
   0202

mov  ax, 124h
push  ax
push   0af8h
push  0eeeh
```




# Pop

- POP copies the contents of the stack pointed to by SP into a register or variable and increments SP. Two registers (CS and IP) cannot be used as operands.
- Allowed forms:
  - pop reg
  - pop memval
- Examples:
  - pop cx
  - pop count ;where count dw ?

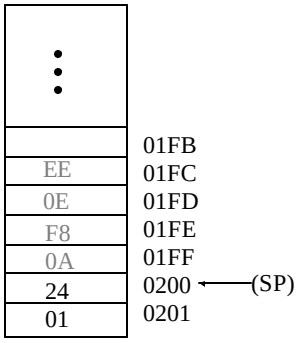
# Pop Examples

```
;initial SP = 01FC

pop      BX
pop      DX

BX = 0EEEh
DX = 0AF8
```

If we were to pop again, the next value popped off would be 0124.



## More Instructions

- PUSHF – pushes the flags register on to the stack
- POPF – restores the flags register from the stack

## Common Stack Uses

## Stack Overflow

## Saving and Restoring Registers

- example p 135, Irvine

## Procedures

- You did this in homework 3 with wrint and readint.
- Some terms:
  - function: a procedure that returns a value
  - subroutine: a procedure (the terms are interchangeable)

## PROC and ENDP

- PROC identifies the start of a procedure
- ENDP identifies the end
- example 1, p. 136 in Irvine

## CALL and RET

- CALL – pushes IP on the stack (recall, IP holds the address of the next instruction), puts the address of the label (subroutine) into IP.
- RET – pops the stack into IP to return to the point at which the subroutine was called.

```
        TITLE   Procedure Calls
;       two calls to a procedure that increments every element
;       of an array

        .model small
        .stack 100h

        .data
List            DW      5FFFh, 0Ah, 12h, 17h
Array           DW      -9, 4, 7, 0, 14, 9
count1  DW      4
count2  DW      6

        .code
        .startup
        mov     bx, offset List     ;first call
        mov     cx, count1
        call    IncProc

        mov     bx, offset Array    ;second call
        mov     cx, count2
        call    IncProc

        nop                         ;can examine arrays with
                                    ;the debugger here
        .exit
        ;return to DOS
```

```
;procedure to increment all the elements in an array of words
;BX contains the base address of the array
;CX contains the number of elements in the array

IncProc  proc
         sub     si, si              ;start index at 0
Lup:     inc     word ptr [bx] [si]  ;use indexed addressing
         add     si, 2
         ;dealing with words, not bytes
         loop    Lup                 ;go to next element
         ret
         ;return - must be used with Call
IncProc  endp

         end                         ;end of assembly
```

## Nested Procedure Calls

- example 4 from text

## Near vs Far Calls

- When the caller and subroutine are in the same segment, the CALL instruction generates code for a NEAR CALL.
- When the caller and subroutine are in different segments, the assembler generates a FAR CALL.

## FAR CALL

- FAR CALL
  - saves both CS and IP on the stack (pushes CS first).
  - loads the subroutine's CS and IP
  - generates a different form of return (RETF) that restores CS and IP from the stack

## Using FAR CALL

- Use FAR:
  - when linking asm routines to HLL programs (some require FAR calls)
  - when calling certain library routines that are set up for far calls
  - when your program size exceeds 64K (medium or large memory models). In this case, there will be multiple code segments, requiring far calls.
- NEAR is default
- NEAR calls are faster

## Far Call, cont.

- example, p. 140 in Irvine