Example: Addressing in Data Segment

Why do we use Data Segment ?

Assume that a program is needed to add 5 bytes of data (25_H, 12_H, 15_H, $1F_H$, and $2B_H$):

Bad programming practice: **(Code & Data are mixed)**

```
MOV AL,00H      ;initialize AL
ADD AL,25H
ADD AL,12H
ADD AL,15H
ADD AL,1FH
ADD AL,2BH      ; AL = 25 + 12 + 15 + 1F + 2B
```

DS:01FF	?
DS:0200	25
DS:0201	12
DS:0202	15
DS:0203	1F
DS:0204	2B
DS:0205	?

Better programming practice:

Assume that the Data segment contains the array of bytes starting from offset address 0200_H.

```
MOV AL,0            ;clear AL
ADD AL,[0200]       ;add the contents of DS:200 to AL
ADD AL,[0201]       ;add the contents of DS:201 to AL
ADD AL,[0202]       ;add the contents of DS:202 to AL
ADD AL,[0203]       ;add the contents of DS:203 to AL
ADD AL,[0204]       ;add the contents of DS:204 to AL
```

Stack

In 8086, the main stack register is called stack pointer - SP. The stack segment register (SS_R) is usually used to store information about the memory segment that stores the call stack of currently executed program. SP points to current stack top.

The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU.

It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack (push) and gets incremented by two as we retrieve a data word from the stack back to the CPU register (pop).

The process of storing the data in the stack is called 'pushing into' the stack and the reverse process of transferring the data back from the stack to the CPU register is known as 'popping off' the stack. The stack is essentially Last-In-First-Out (LIFO) data segment. This means that the data which is pushed into the stack last will be on top of stack and will be popped off the stack first.

The stack pointer is a 16-bit register that contains the offset address of the memory location in the stack segment. The stack segment, like any other segment, may have a memory block of a maximum of 64 Kbytes locations, and thus may overlap with any other segments. Stack Segment register (SS) contains the base address of the stack segment in the memory.

If the stack top points to a memory location 52050H, it means that the location 52050H is already occupied with the previously pushed data. The next 16-bit push operation will decrement the stack pointer by two, so that it will point to the new stack-top 5204EH and the decremented contents of SP will be 204EH. This location will now be occupied by the recently pushed data.

Thus for a selected value of SS, the *maximum* value of SP=FFFFH and the segment can have a maximum of 64K locations. If the SP starts with an initial value of FFFFH, it will be decremented by two whenever a 16-bit data is pushed onto the stack. After successive push operations, when the stack pointer contains 0000H, any attempt to further push the data to the stack will result in a *stack overflow*.

There are many real-life *examples* of a stack, such as backtracking algorithms, a stack of dinner plates, a box of pringles potato chips. The basic operating principle is that the last item you put in is the first item you can take out.

Stack is an area of memory for *keeping temporary data*. Stack also is used by CALL instruction to keep return address for procedure, RET instruction gets this value from the stack and returns to that offset.

Quite the same thinks happens when INT instruction calls an interrupt, it stores in stack flag register, code segment and offset, RET instruction is used to return from interrupt call.

We can also use the stack to keep any other data, there are two instructions work with the stack:

1. **PUSH instruction - stores 16-bit value in the stack.**
2. **POP instruction- gets 16-bit value from the stack.**

1. ***Syntax for PUSH instruction: PUSH REG:***

PUSH REG

PUSH SREG

PUSH MEMORY

PUSH Immediate

REG: **AX, BX, CX, DX, SI, DI, BP, SP.**

SREG: **DS,CS, ES, SS.**

MEMORY: **[BX], [BX]+SI+7, etc…**

Immediate: **5, -24, 3Fh, 10001101b, etc…**

2. ***Syntax for PUSH instruction: POP REG:***

POP REG

POP SREG

POP MEMORY

REG: **AX, BX, CX, DX, SI, DI, SP, SP.**

SREG: **DS, ES, SS, (Except CS).**

MEMORY: **[BX], [BX]+SI+7, etc…**

Notes:

- ***PUSH and POP work with 16-bit values only.***
- ***PUSH IMMEDIATE works only on 80186 CPU and later!***
- ***The stack uses FILO (First In Last Out) algorithm.***
- ***PUSH AX, store the register AX to the stack, at first stores the Low (AL) and then stored the High (AH).***
- ***POP AX, get the last value stored in the stack to the register AX, at first, the last value is stored in the High (AH) and then other value is stored in the Low (AL).***

Example 1:

You have the values (1, 2, 3, 4 ,5)$_H$ in the stack, and the value of the stack segment (SS) is 1000$_H$ and the physical of the last values which stored in the stack (5) is 11000$_H$. Show how can you pop the number from the stack? And what is the stack pointer in each case?

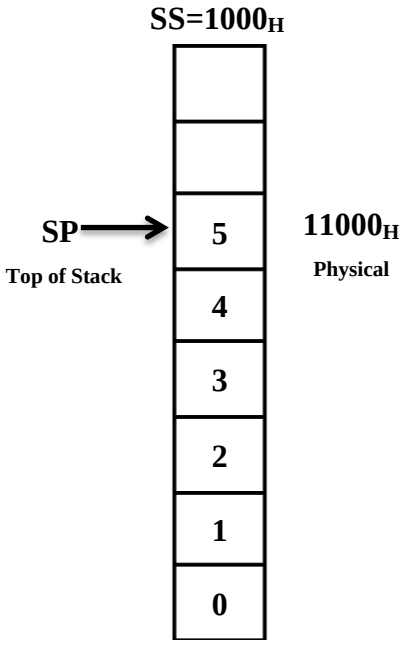
1. The last SP = Physical address – SS*10$_H$

= 11000 - (1000*10) = 1000$_H$

POP AX

AX = 0504$_H$

05Ah	04Al



2. Other SP = 1000 + 2 = 1002$_H$

POP BX

BX = 0302$_H$

3. Other SP = 1002 + 2 =1004$_H$

POP CX

CX = 0100$_H$

4. Final SP = 1004 + 2 =1006$_H$

It is very important to do an equal number of PUSHs and POPs, otherwise, the stack may be corrupted and it will be impossible to return to the operating system. As you already know we use RET instruction to return to the operating system, so when the program starts there is a return address in the stack (generally it's 0000$_H$).

PUSH and POP instructions are especially useful because we don't have too many registers to operate with, so here is a trick:

- **Store the original value of the register in the stack (Using PUSH).**
- **Use the register for any purpose.**
- **Restore the original value of the register from the stack (Using POP).**

The stack memory area is set by SS (Stack Segment) register, and SP (Stack Pointer) register. Generally, the CPU *sets values* of these registers on program start.

Example 2:

```
ORG  100h
     Mov   AX, 1234h
     Push   AX          ; store value of AX in stack.
     Mov   AX, 5678h    ; modify the AX value.
     Pop    AX          ; restore the original value of AX. (From Stack).
     RET
END
```

Example 3:

Another use of the stack is for exchanging the values.

```
ORG  100h
     Mov   AX, 1212h   ; store 1212h in AX.
     Mov   BX, 3434h   ; store 3434h in BX
     Push  AX          ; store value of AX in stack.
     Push  BX          ; store value of BX in stack.
     Pop   AX          ; set AX to original value of BX.
     Pop   BX          ; set BX to original value of AX.
     RET
END
```

The exchange happens because the stack uses the LIFO (Last In First Out) algorithm, so when we push 1212h and then 3434h, on pop we will first get 3434h and only after it 1212h.

"SP" points to the current stack top.

"PUSH instruction" → **SP = SP - 2**

"POP instruction" → **SP = SP + 2**

Home Work:

1. **You have AX=1212h and BX=3434h write an 8086 program to exchange between the above two registers by using the stack.**
2. **Write an 8086 program to calculate the expression C=A*B, where:**

A is the value stored in the memory at the physical address 12345h and 12346h, and

B is stored also in memory at the physical address 10000h and 10001h, store the result in the stack.

3. **Find the value of the SS and SP after run the above program and then calculate the physical address of the stack.**