Extra segment (ES)

ES is a segment register used as an extra data segment. Although in many normal programs this segment is not used, its use is absolutely essential for string operations and is discussed in detail in later chapters.

Memory map of the IBM PC

For a program to be executed on the PC, DOS must first load it into RAM. **Where in RAM will it be loaded?**

To answer that question, we must first explain some very important concepts concerning memory in the Pc. The 20-bit address of the 8088/86 allows a total of 1 megabyte (1024K bytes) of memory space with the address range 00000 - FFFFF.

During the design phase of the first IBM PC, engineers had to decide on the allocation of the 1-megabyte memory space to various sections of the PC. This memory allocation is called a memory map. The memory map of the IBM PC is shown in Figure 1-3. Of this 1 megabyte, 640K bytes from addresses 00000 - 9FFFFH were set aside for RAM. The 128K bytes from AOOOOH to BFFFFH were allocated for video memory. The remaining 256K bytes from COOOOH to FFFFFH were set aside for ROM.

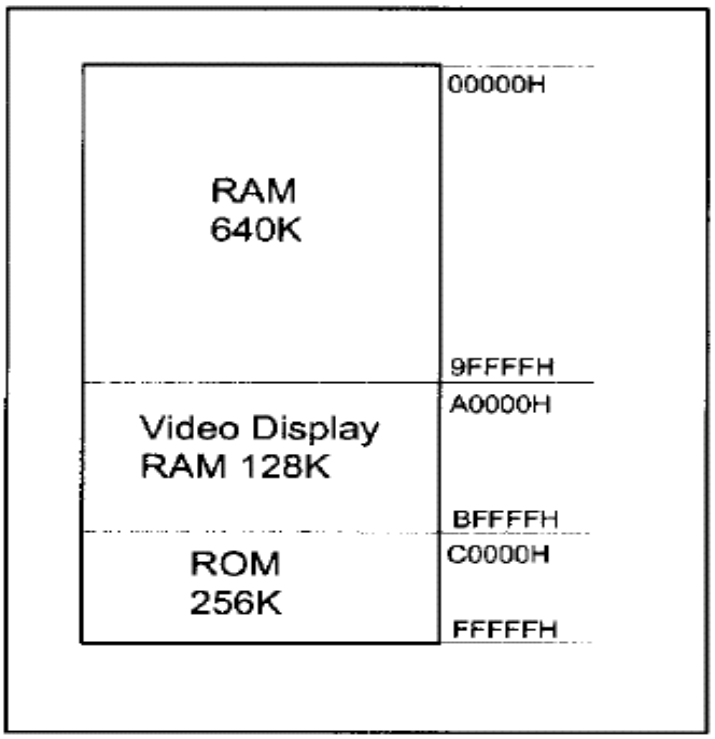


Figure 1-3. Memory Allocation in the PC

1.5 What is a stack, and why is it needed?

The stack is a section of read/write memory (RAM) used by the CPU to store information temporarily. The CPU needs this storage area since there are only a limited number of registers

How stacks are accessed

If the stack is a section of RAM, there must be registers inside the CPU to point to it. The two main registers used to access the stack are the SS (stack segment) register and the SP (stack pointer) register. These registers must be loaded before any instructions accessing the stack are used.

Every register inside the 80x86 (except segment registers and SP) can be stored in the stack and brought back into the CPU from the stack memory. The storing of a CPU register in the stack is called a push, and loading the contents of the stack into the CPU register is called a pop. In other words, a register is pushed onto the stack to store it and popped off the stack to retrieve it. The job of the SP is very critical when push and pop are performed.

In the 80x86, the stack pointer register (SP) points at the current memory location used for the top of the stack and as data is pushed onto the stack it is decremented. It is incremented as data is popped off the stack into the CPU.

Pushing onto the stack

Notice in Example 1-6 that as each PUSH is executed, the contents of the register are saved on the stack and SP is decremented by 2. For every byte of data saved on the stack, SP is decremented once, and since push is saving the contents of a 16-bit register, it is decremented twice. Notice also how the data is stored on the stack.

In the 80x86, the lower byte is always stored in the memory location with the lower address. That is the reason that 24H, the contents of AH, is saved in memory location with address 1235 and AL in location 1234.

Popping the stack

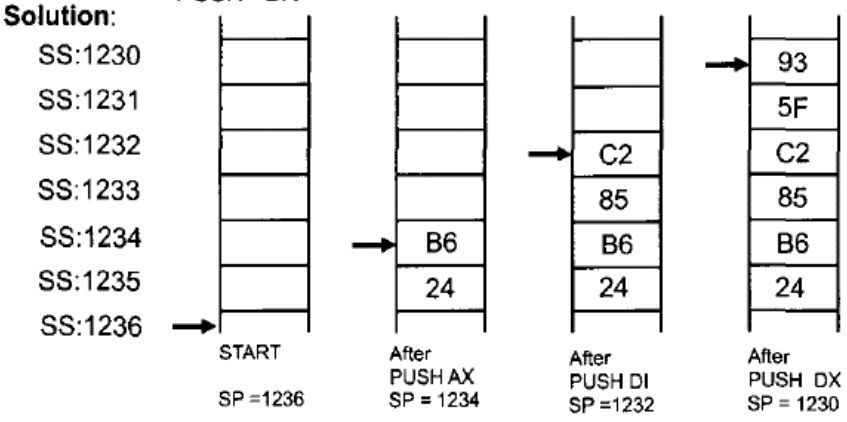
Popping the contents of the stack back into the 80x86 CPU is the opposite process of pushing. With every pop, the top 2 bytes of the stack are copied to the register specified by the instruction and the stack pointer is incremented twice. Although the data actually remains in memory, it is not accessible since the stack pointer is beyond that point. **Example 1-7** demonstrates the POP instruction

Example 1-6

Assuming that SP = 1236, AX = 24B6, DI = 85C2, and DX = 5F93, show the contents of the stack as each of the following instructions is executed:

```
PUSH AX
PUSH DI
PUSH DX
```

Solution:

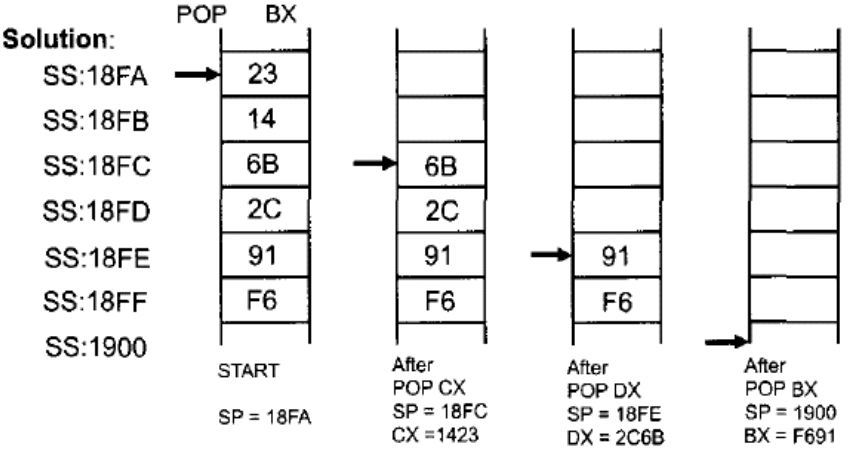


Example 1-7

Assuming that the stack is as shown below, and SP = 18FA, show the contents of the stack and registers as each of the following instructions is executed:

```
POP CX
POP DX
POP BX
```

Solution:



Logical address VS, physical address for the stack:

Now one might ask, **what is the exact physical location of the stack?** That depends on the value of the stack segment (SS) register and SP, the stack pointer. To compute physical addresses for the stack, the same principle is applied as was used for the code and data segments. The method is to shift left SS and then add offset SP, the stack pointer register. This is demonstrated in Example 1-8.