Memory segmentation

Main memory model

Instructions and data are stored in main memory.

The (main) memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0. These cells are organized in the form of groups of fixed number of cells.

An entity consisting of 8 bits is called a byte, of 16 bits – a **word,** of 32 bits – a **double word.**

In order to be able to move a byte in and out of the memory, a distinct address has to be assigned to each byte.

The number of bits, *l*, needed to distinctly address M bytes in a memory is given by

$$l = \log_2 M$$

If the size of the memory is 1 MB, then the number of bits in the address is

$$\log_2(2^{20}) = 20 \text{ bits.}$$

	7 0
FFFFF	
FFFFE	
FFFFD	
.....	
.....	
10000	
0FFFF	
.....	
00001	
00000	

The addressable memory of I8086 contains 2^{20} bytes (1 Mb). The physical addresses are within the range 00000-FFFFFh.

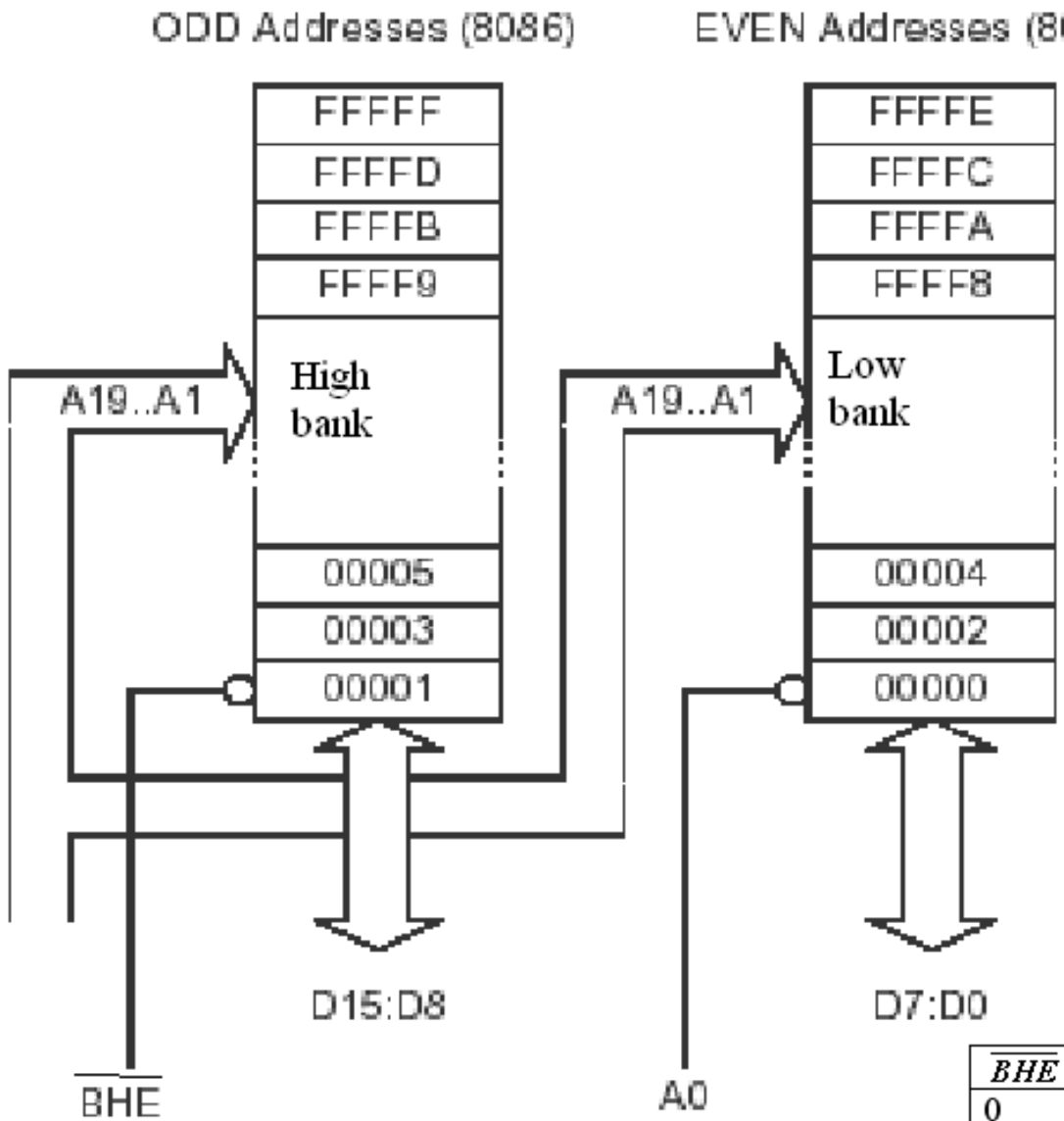
Locations 0H-7FH (128 bytes) and FFFF0-FFFFF (16 bytes) are reserved for special use (interrupts and system start after reset)

		22 H		Unaligned
		21 H		DW
		20 h		DB
24B H	46	1F H		Aligned
24A H	00	1E H		DW
249 H	65	1D H		DB
248 H	3A	1C H		DB
247 H	8C	1B H		Instruction
246 H	04	1A H		
		19 H		Instruction

Any 2 neighbour bytes can store a word (16 bits). The **smaller address** contains the **smaller byte**. The **address of the word** is the address of its **smaller byte**.

This strategy to store data is called Little Endian (the opposite strategy is called Big Endian and it applied by Motorola, Spark and most RISC machines).

The word with even address is called aligned. The word with odd address is called unaligned. The processor transfer words with even addresses in **1 memory access cycle** and words with odd addresses in 2 cycles.



$\overline{BHE}$	A0	Data
0	0	Word
0	1	Upper byte (odd address)
1	0	Lower byte (even address)
1	1	none

Memory segmentation

Segmentation provides a powerful memory management mechanism:

It allows programmers to partition their programs into modules that operate independently of one another.

Segments provide a way to easily implement object-oriented programs.

Segments allow two processes to easily share data.

It allows extending the addressability of a processor. In the case of the 8086, segmentation let Intel's designers extend the maximum addressable memory from 64KB to 1MB.

- **Disadvantage:** Difficulties with physical address manipulation in programs.

A full segmented address contains a segment component and an offset component

segment:offset.

On the 8086 through the 80286, these two values are 16 bit constants. On the 80386 and later, the offset can be a 16 bit constant or a 32 bit constant.

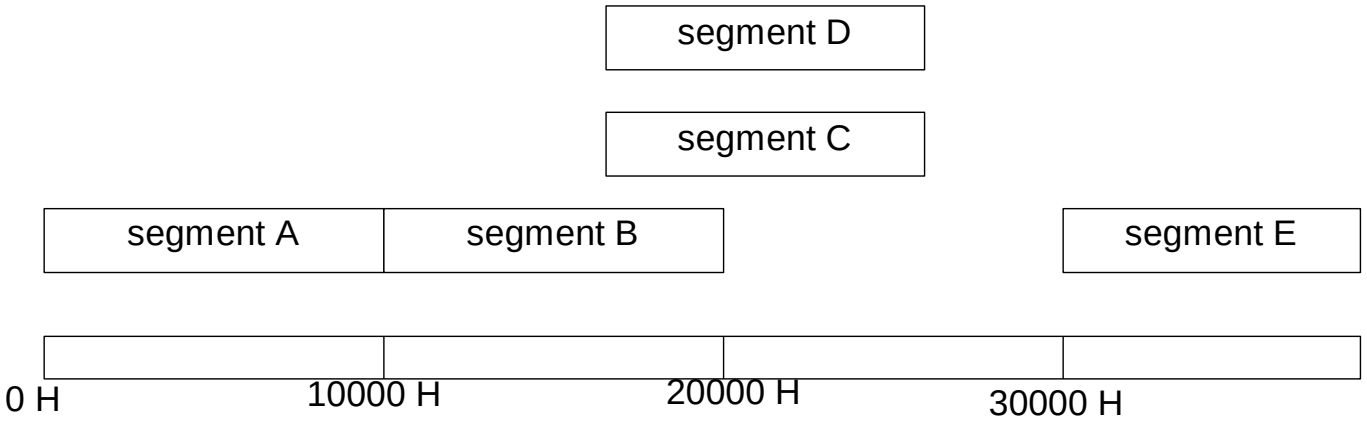
The size of the offset limits the maximum size of a segment.

On the 8086 with 16 bit offsets, a segment may be no longer than $2^{16}=2^6*2^{10}$=64KB; The 80386 and later processors allow 32 bit offsets with segments as large as $2^{32}=2^2*2^{30}$=4GB.

The segment portion is 16 bits on all 80x86 processors. This lets a single program have up to 65,536 different segments in the program.

All memory space is considered as a set of 64 Kbyte size segments. The segments are defined for each application. Segments are considered to be independent and uniquely addressable. For each program can be currently addressed 4 segments using CS, DS, ES and SS.

Segment registers are initialised at the beginning of the application. They contain the base (low) address of the segment which is always a multiple of 16 (4 low bits are considered 0).



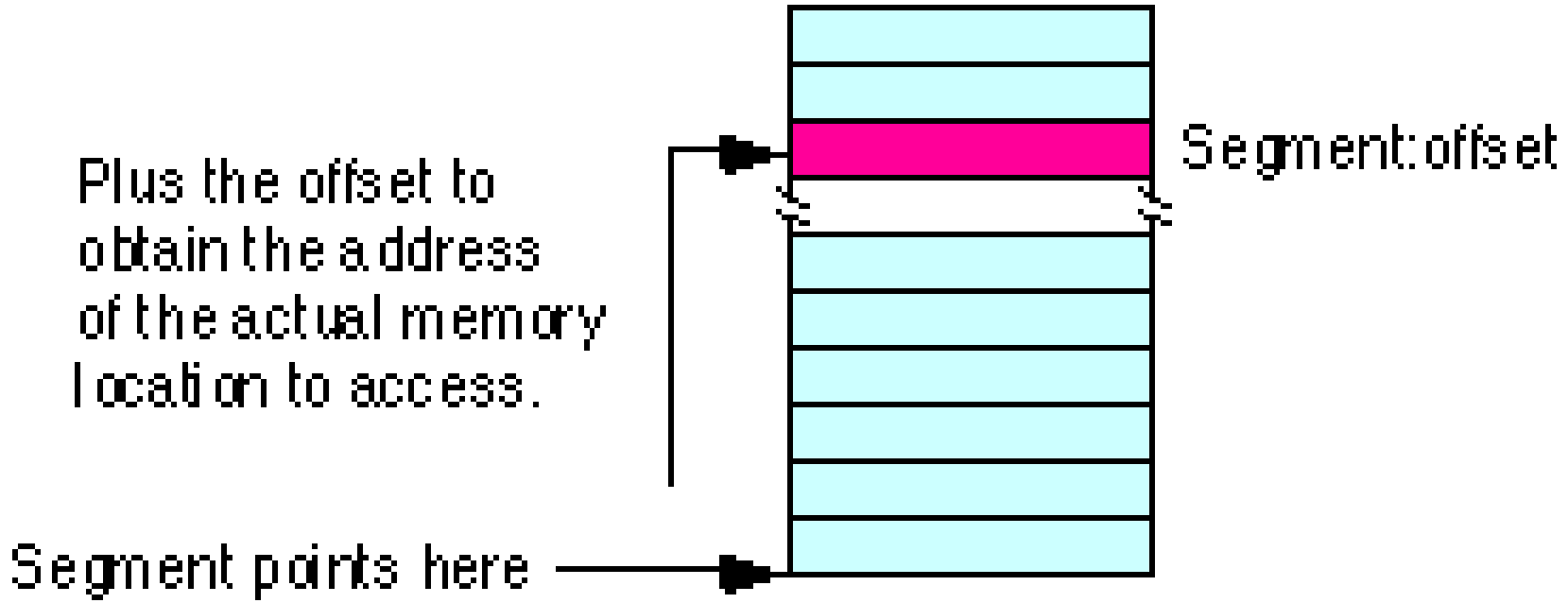
Physical address calculation

Addresses in the programs - **logical addresses**.

The linear address that appears on the address bus - **physical address.**

Logical address notation

segment: offset


Segment:offset
Plus the offset to
obtain the address
of the actual memory
location to access.
Segment points here

Physical address calculation ***segment*10H+offset***

Segment*10H is equivalent to 1 hexadecimal (4 bits) shift left. To calculate the physical address in BIU the base address is shift 4 bits left and the offset is added.

Example. If (CS)=123A h and (IP)=341B h, the physical address will be

$$\begin{array}{ll} 123A0 & \textit{the base address of the segment} \\ \quad 341B & \textit{offset} \\ \hline 157BB & \textit{a physical address} \end{array}$$

Sources of physical address:

Type of memory access	Implicit segment	Alternative segment	Offset
Instruction fetch	CS	-	IP
Stack operation	SS	-	SP
Variable	DS	CS, ES, SS	EA
String source	DS	CS, ES, SS	SI
String destination	ES	-	DI
BP as base Rg	SS	CS, DS, ES	EA

Stack memory

A stack memory is a small area of reserved memory used in the following cases:

1. To store temporary the data from general purpose registers;
2. To store the content of PSW, CS and IP when an interrupt or a procedure is processed:
3. To transmit the procedures parameters.

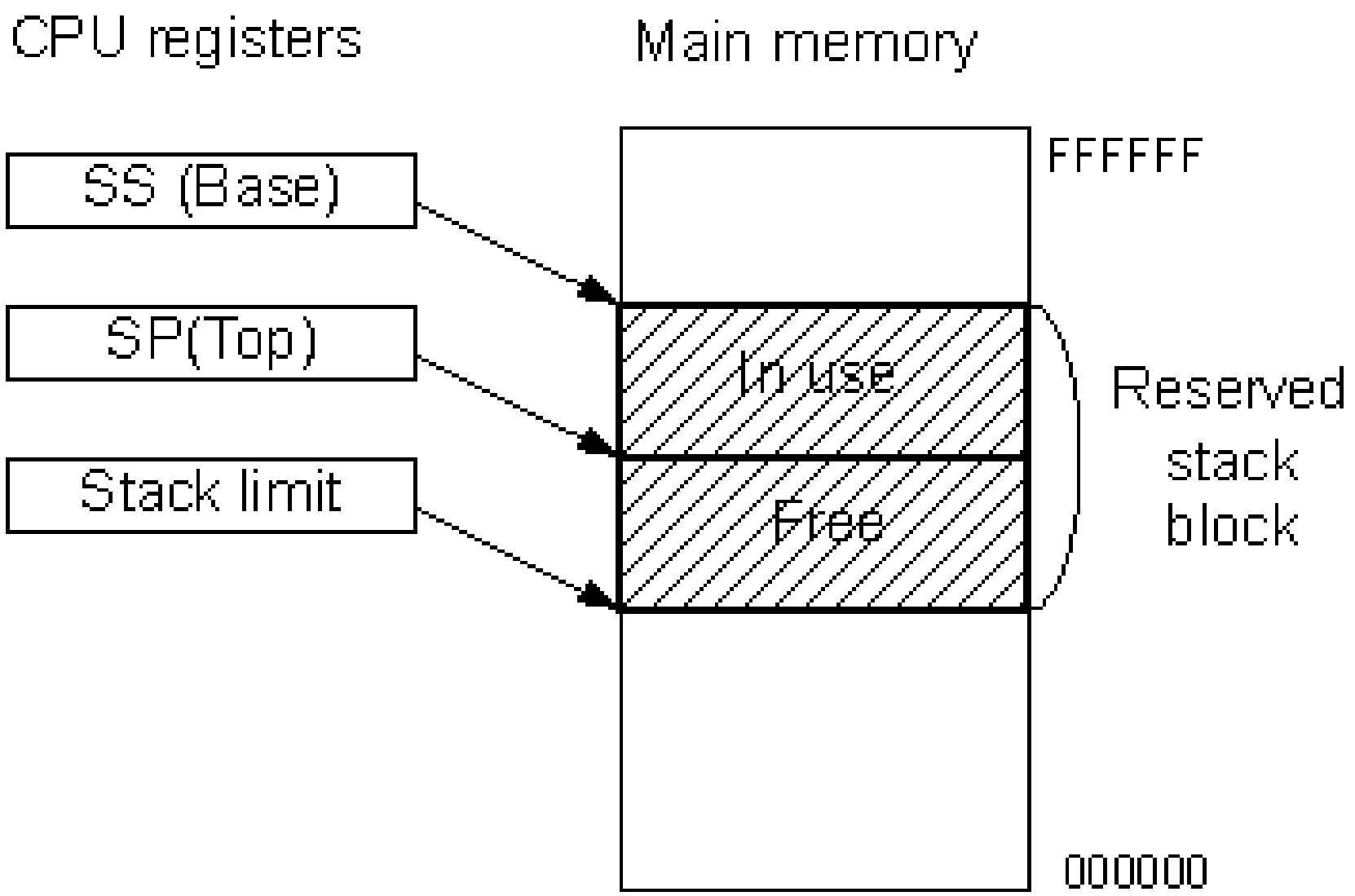
The stack organization principle is **LIFO.**

Stack location is determined by **SS:SP**.

SS holds the base address of stack and **SP holds the offset** of the top of the stack (the most recent stack entry).

Instructions to operate with stack are:

- **PUSH** - Copy specified word to top of the stack.
- **POP** - Copy word from top of the stack to specific location.


CPU registers
Main memory
SS (Base)
SP(Top)
Stack limit
FFFFFF
In use
Free
Reserved
stack
block
000000

According to Intel convention the stack grows from higher addresses to lower addresses

(according to Motorola convention the stack grows from lower addresses to higher addresses).

The base of the stack (SS) is at the high address end of the reserved stack block and the limit is at the low address end.

If all stack elements are 16-bit words (2 bytes), instruction **PUSH** will cause the **decrement of SP with 2** and **POP** will cause the **increment of SP with 2.**

1. Calculate the physical address according to the following logical addresses:

a) 1205H : 709H,

b) ABCDH : 89ABH,

c) FFF0H : 0FFH,

d) 3333H : 4444H,

e) 8000H : 8000H.

2. Calculate the offset according to the following physical addresses (CS=2000H) :

a) 20002H,

b) 20010H,

c) 20300H,

d) 24000H,

e) 2FFFFH.

3. Calculate CS according to the following physical addresses (offset is 400H) :

a) 10400H,
b) B0400H,
c) 30800H,
d) CDE00H,
e) FFFF0H.

4. Which of the following physical addresses belong to the segment with CS=2400H:

a) 33FFFH,
b) 23000H,
c) 27890H,
d) 33000H,
e) 34000H.

5. Physical address of the variable is 358BC H when CS=3234 H. Calculate the physical address of the variable when CS is changing 4310 H.

Load in a variable in memory its address as segment:offset

```
.model small
.data
x dw 1Ah, 1Bh, 1Ch
y  dd ?
.code
 start: mov ax,@data
        mov ds,ax
        mov bx, offset y
        mov cx, seg y
        mov word ptr y, bx
        mov word ptr y+2, cx
        end start
```