

# Computer Architecture & Networks

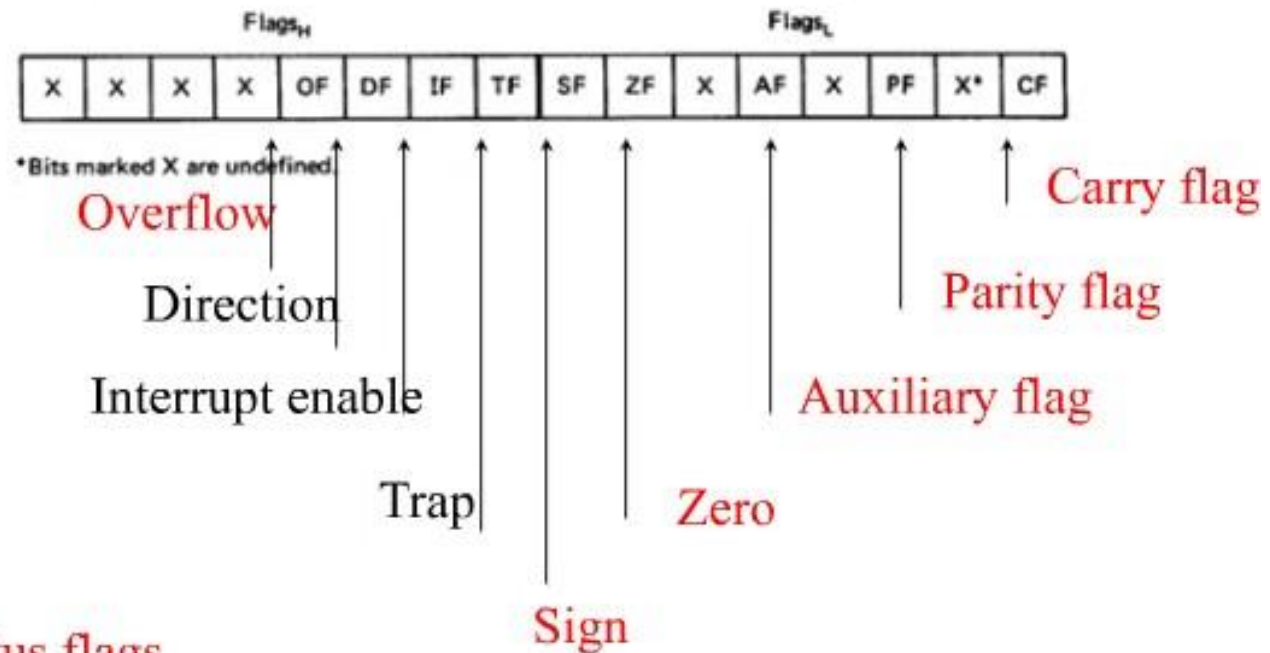
N-Bit Microprocessors

# Flags Register

- Flags indicate the condition of the microprocessor as well as its operation
- The flag bits change after many arithmetic and logic instructions execute
- Example flags,
  - **CF(carry)** indicates carry after addition or a borrow after subtraction
  - **OF(overflow)** is a condition that can occur when signed numbers are added or subtracted
  - **ZF(zero)** indicates that the result of an arithmetic or logic operation is zero (1 = yes)

# Flags Register

## Flags



6 are status flags

3 are control flag

# Flags Register

- DF(direction) indicates left or right for moving or comparing string data.
- IF(interrupt) indicates whether external interrupts are being processed or ignored.
- TF(trap) permits operation of the processor in single step mode.
- SF(sign) contains the resulting sign of an arithmetic operation (1 = negative)
- AF(auxiliary carry) contains carry out of bit 3 into bit 4 for specialised arithmetic.
- PF(parity) indicates the number of 1 bits that result from an operation.

# Self-Test 1: Flags Condition

Assume the following register conditions:

AX = AB00H, BX = 200CH, CX = 3003H, DX = DEFFH

Find the status of the CF, PF, AF, ZF and SF for the following operations:

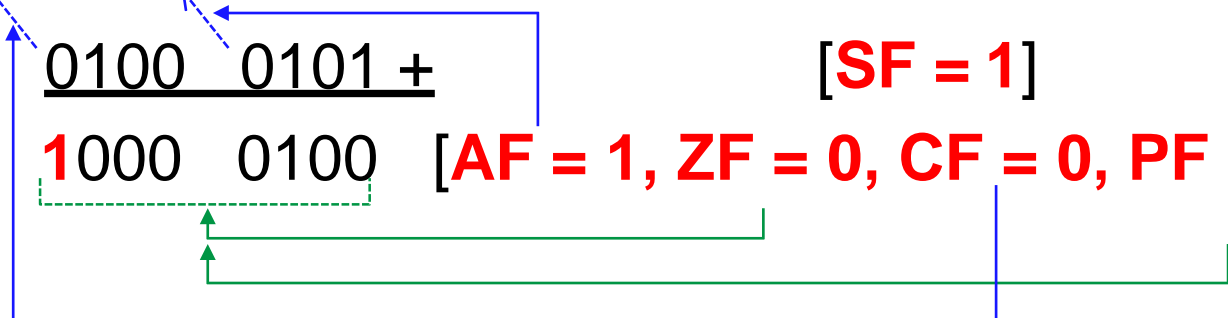
1. MOV BX, 3FH  
ADD BL, 45H
2. ADD BX, DX
3. CMP AX, DX
4. NOT DX

# Self-Test 1: Answers

1. **MOV BX, 3FH**  
**ADD BL, 45H**

BL = 84H (3FH + 45H)

3FH: 0011 1111  
45H: 0100 0101 +  
1000 0100 [SF = 1]  
[AF = 1, ZF = 0, CF = 0, PF = 1]



2. **ADD BX, DX**

200CH:	0010 0000	0000	1100
DEFFH:	1101 1110	1111	1111 +
	<u>1111 1111</u>	0000	1011 (FF0BH)

[AF = 1, ZF = 0, CF = 0, PF = 0, SF = 1]

# Self-Test 1: Answers (cont)

## 3. **CMP AX, DX**

AB00H: 1010 1011 0000 0000

DEFFH: 1101 1110 1111 1111 –

1100 1100 0000 0001 (CC01H)

[**AF = 1, ZF = 0, CF = 1, PF = 0, SF = 1**]

## 4. **NOT DX**

DEFFH: 1101 1110 1111 1111

NOT: 0010 0001 0000 0000 (2100H)

[**AF = 0, ZF = 0, CF = 0, PF = 0, SF = 0**]

# Self-Test 2: Arithmetic & Logical

Assume the following register conditions:

CF = 1, AX = ABCDH, BX = EF01H, DX = 2345H

Perform the following operations. Indicate the result and the register where it is stored. The operations are independent of each other.

1. XOR AL, 79H
2. MOV CL, 4H  
ROR DX, CL
3. OR BX, DX
4. AND AX, BX
5. MOV CL, 3H  
RCL BH, CL



# Self-Test 2: Answers

1. **XOR AL, 79H**

CDH: 1100 1101

79H: 0111 1001  $\oplus$

1011 0100 [AL = **B4H**]

2. **MOV CL, 4H**

**ROR DX, CL**

2345H: 0010 0011 0100 0101

After 4 rotations: 0101 0010 0011 0100  $\Rightarrow$  **5234H** =  
~~DX~~

3. **OR BX, DX**  
 EF01H: 1110 1111 0000 0001

2345H: 0010 0011 0100 0101

1110 1111 0100 0101  $\Rightarrow$  **EF45H** =

~~DX~~

# Self-Test 2: Answers (cont)

## 4. **AND AX, BX**

ABCDH: 1010 1011 1100 1101

EF01H: 1110 1111 0000 0001

1010 1011 0000 0001  $\Rightarrow$  **AB01H**

## 5. **MOV ~~AX~~ CL, 3H**

**RCL BH, CL**      **CF = 1**

EFH: 1110 1111

After 1<sup>st</sup> rotation: 1101 1111, CF = 1

After 2<sup>nd</sup> rotation: 1011 1111, CF = 1

After 3<sup>rd</sup> rotation: 0111 1111  $\Rightarrow$  **7FH = BH, CF = 1**

# Self-Test 3: Physical Address

The register content for an Intel 8086 microprocessor is as follows:

CS = 1000H, DS = 2000H, SS = 3000H, SI = 4000H, DI = 5000H  
BX = 6080H, BP = 7000H, AX = 25FFH, CX = 8791H, DX = 1299H

Calculate the physical address of the memory where the operand is stored and the contents of the memory locations in each of the addresses shown below:

1. MOV [SI], AL
2. MOV [DI+6H], BX
3. MOV [SI+BX-8H], AX
4. MOV [DI][BX]+28H, CX
5. MOV [BP][SI]+10H, DX




# Computer Architecture & Networks

N-Bit Microprocessors



# Content

- Memory Address Space and Data Organization
- Data Types
- Segment Registers and Memory Segmentation

- 
- In computer architecture, a register is a small amount of storage (memory) available on the CPU whose contents can be accessed more quickly than storage available elsewhere (cache, RAM, etc.).



# Software Model of the 8088/8086 Microprocessor

- The purpose of developing a software model is to help the programmer to understand the operation of microprocessor in software point of view.
- Thus, hardware architectural is not necessarily need to know. But it is important to know various registers, external memory and I/O peripherals.
- The figure illustrates the software architecture of 8088 microprocessor.

The 8088 microprocessor includes 13 16 bit internal registers:

Instruction pointer (IP)

Four data registers  
(AX, BX, CX, DX)

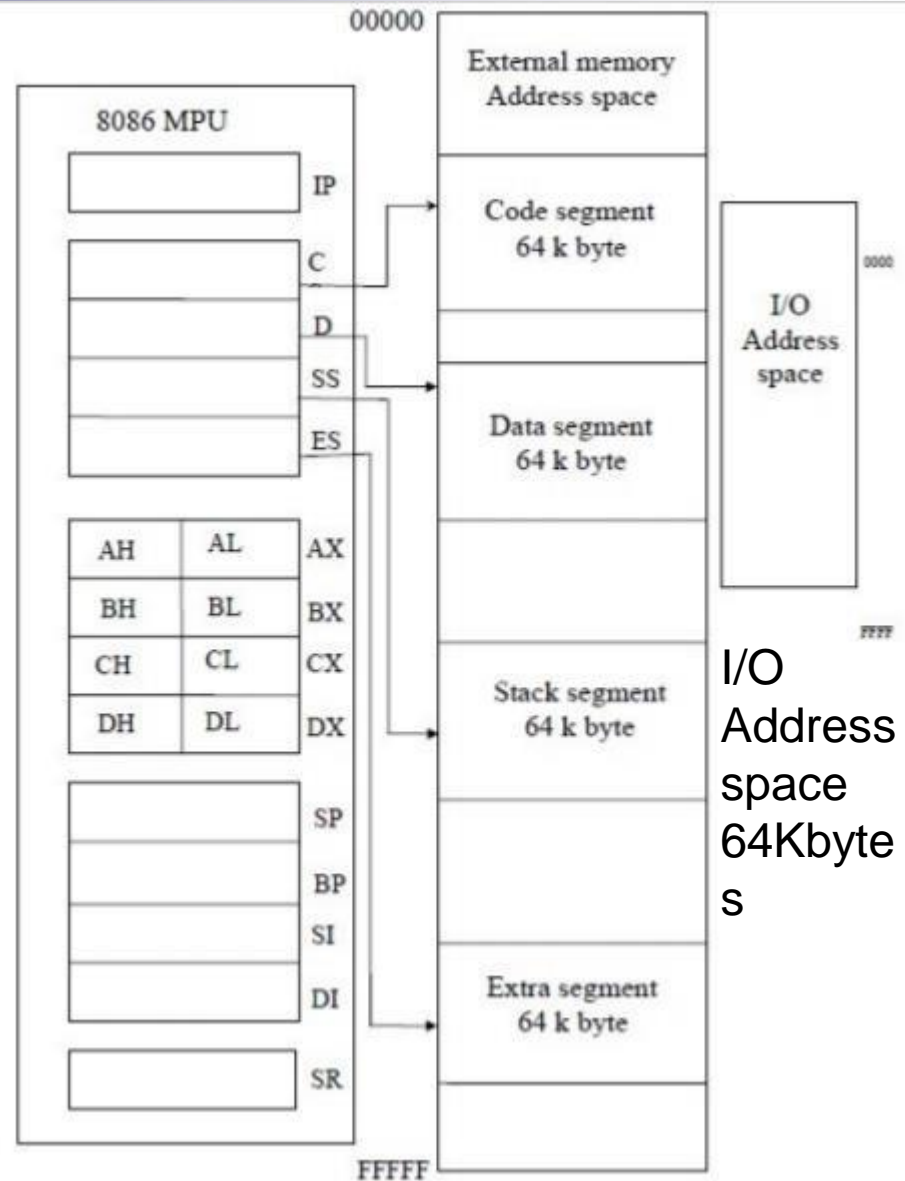
Two pointer registers  
(BP and SP)

Two index registers  
(SI and DI)

Four segment registers  
(CS, DS, SS and ES)

Status register (SR)

Independent memory  
address space  
1 Mbytes



Software model of the 8086/8088 microprocessor



# Memory Address Space and Data Organization

- Lets see how information stored in memory.
- Individual bytes of data stored at consecutive addresses over the address range  $00000_{16}$  to  $FFFFF_{16}$ .
- Thus, memory organized as 8 bit bytes not as 16 bits words.
- The 8088 can access any two consecutive bytes as a word of data. The lower addressed byte is the least significant byte of word, and the higher-addressed byte is its most significant byte.

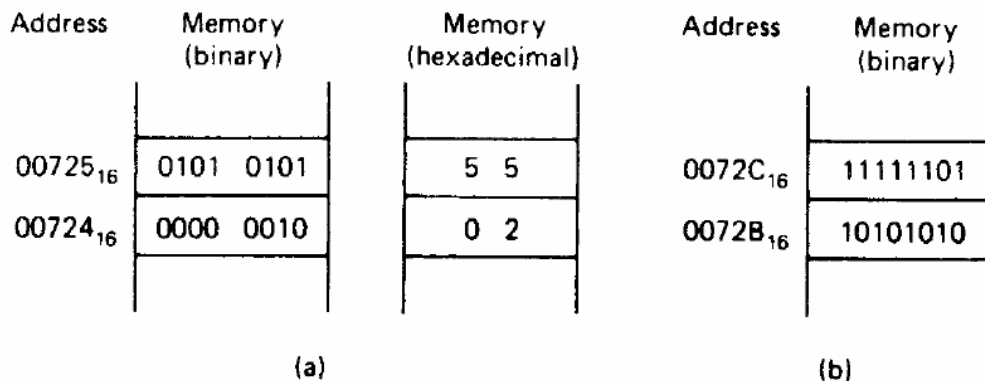
# Memory Address Space and Data Organization

FFFFFF
FFFFFE
FFFFFD
FFFFFC
3
2
1
0

Memory address space

# Memory Address Space and Data Organization

- Lower address byte and higher address byte



- How a word of data is stored in memory for Figure (a)?
- The two bytes represent the word, Higher-add storage + Lower address storage

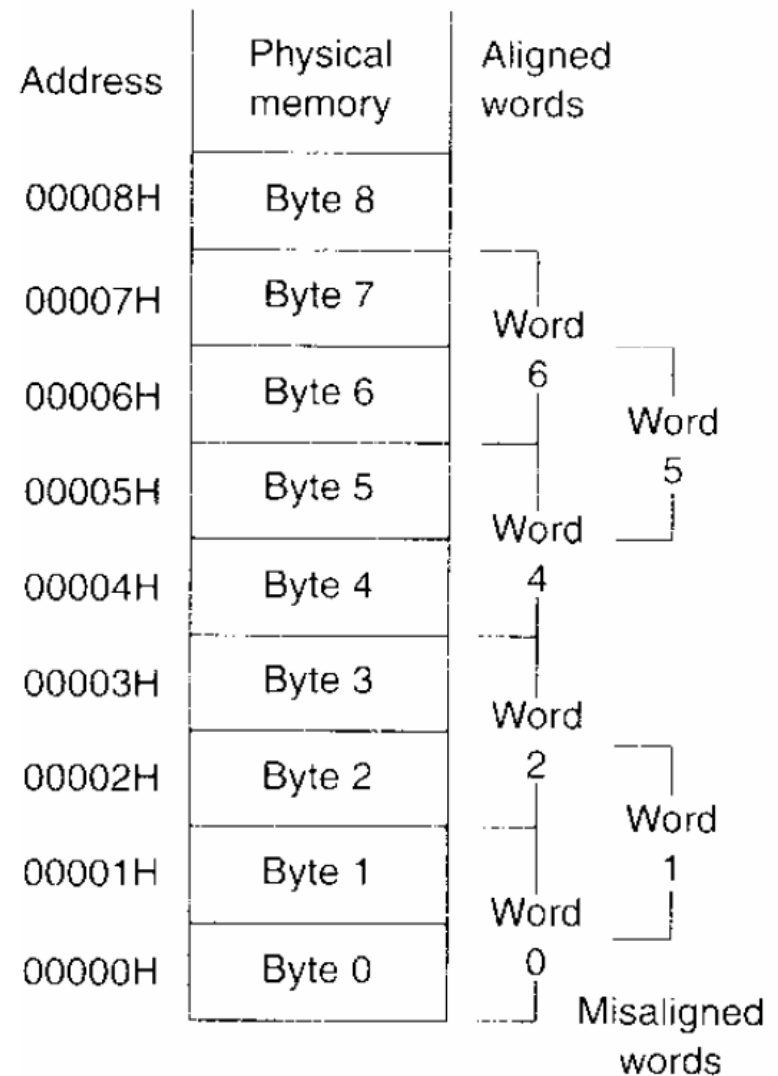
$$0101\ 0101\ 0000\ 0010_2 = 5502_{16}$$



# Memory Address Space and Data Organization

- To permit efficient use of memory, words are stored at even- or odd-address boundaries (memory bank)
  - The LSB of the address determines the type of word boundary
  - '0' : even address
  - 2 consecutive bytes with lower byte at an even address
- Bytes and words at even-addressed boundaries can be accessed by 1 bus cycle
- Bytes and words at odd-addressed boundaries require 2 bus cycles to access
- A double word is another data form that can be processed by the 8088 microcomputer. A double word corresponds to four consecutive bytes of data stored in memory.

- **Aligned word or misaligned word**
- A word of data stored at an even address boundary –  $00000_{16}$ ,  $00002_{16}$ ,  $00004_{16}$ ... aligned word
- A word of data stored at an odd address boundary –  $00001_{16}$ ,  $00003_{16}$ ,  $00005_{16}$ ... misaligned word
- A double word corresponds to four consecutive bytes of data stored in memory.
- Note\* When expressing addresses and data in hexadecimal form it is common to use the letter H to specify the base.



# Memory Address Space and Data Organization

- EXAMPLE:
- What is the data shown in the Figure (b)?
- Express the result in hexadecimal form. Is it stored at an even or odd addressed word boundary? Is it aligned or misaligned word of data?

Address	Memory (binary)	Memory (hexadecimal)
00725 <sub>16</sub>	0101 0101	5 5
00724 <sub>16</sub>	0000 0010	0 2

(a)

Address	Memory (binary)
0072C <sub>16</sub>	11111101
0072B <sub>16</sub>	10101010

(b)

# Memory Address Space and Data Organization

## ■ SOLUTION:

$$11111101_2 = FD_{16} = FDH$$

$$10101010_2 = AA_{16} = AAH$$

Together the two bytes give the word

$$1111110110101010_2 = FDAA_{16} = FDAAH$$

Expressing the address of the least significant byte in binary form

$$\text{gives } 0072BH = 0072B_{16} = 0000\ 0000\ 0111\ 0010\ 1011_2$$

The rightmost bit(LSB) is logic 1. Therefore, it is misaligned word of data.

Address	Memory (binary)	Memory (hexadecimal)
00725 <sub>16</sub>	0101 0101	5 5
00724 <sub>16</sub>	0000 0010	0 2

(a)

Address	Memory (binary)
0072C <sub>16</sub>	11111101
0072B <sub>16</sub>	10101010

(b)

# Memory Address Space and Data Organization

- 8086 processor has a 20-bit address bus, thus can address up to  $2^{20}$  or 1,048,576 memory locations. Since each memory location stores one byte, 8086 systems support up to **1,048,576 bytes** or **1Mbyte** of memory.
- The range of the memory address are from 0000 0000 0000 0000 0000 to 1111 1111 1111 1111 1111. To simply it, we normally write the address in hexadecimal : 00000H – FFFFFH.
- The memory subsystem can access one or two bytes, where any 2 consecutive bytes can be accessed as a word
  - The lower byte is less significant while the higher byte is more significant (Little Endian Byte Ordering).



# Memory Address Space and Data Organization

- A pointer is a double word (multiple of 4) used to access data or code in memory. The higher address word represents the *segment base address* while the lower address word represents the *offset*.

Address	Memory (binary)	Memory (hexadecimal)	Address	Memory (hexadecimal)
00007 <sub>16</sub>	0011 1011	3 B	0000B <sub>16</sub>	A0
00006 <sub>16</sub>	0100 1100	4 C	0000A <sub>16</sub>	00
00005 <sub>16</sub>	0000 0000	0 0	00009 <sub>16</sub>	55
00004 <sub>16</sub>	0110 0101	6 5	00008 <sub>16</sub>	FF

# Memory Address Space and Data Organization

*Example: Storage of a pointer in memory*

*Combine both MSB [00007H] + [LSB 00006H]*

Higher-addressed word :

Segment base address =

$3B4C_{16} = 0011101101001100_2$

Lower- addressed word :

Offset value =

$0065_{16} = 0000000001100101_2$

The complete double word is  $3B4C0065_{16}$ . Since the double word is starts at address  $00004_{16}$ , is an example of aligned double word of data.

Address	Memory (binary)	Memory (hexadecimal)	Address	Memory (hexadecimal)
$00007_{16}$	0011 1011	3 B	$0000B_{16}$	A0
$00006_{16}$	0100 1100	4 C	$0000A_{16}$	00
$00005_{16}$	0000 0000	0 0	$00009_{16}$	55
$00004_{16}$	0110 0101	6 5	$00008_{16}$	FF



# SELF TEST

- How should the pointer with segment base address equal to A000H and offset address 55FFH be stored at an even-address boundary starting at 00008H? Is the double word aligned or misaligned?

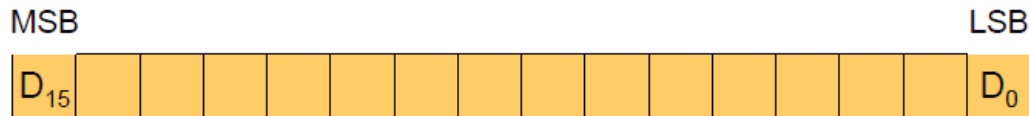
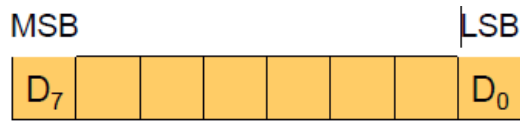
# SELF TEST

- How should the pointer with segment base address equal to A000H and offset address 55FFH be stored at an even-address boundary starting at 00008H? Is the double word aligned or misaligned?
- Solution:
- Storage of the two-word pointer requires four consecutive byte locations in memory, starting at address 00008<sub>16</sub>. The least-significant byte of the offset is stored at address 00008<sub>16</sub> and is shown as FF<sub>16</sub> in the previous figure. The most significant byte of the offset, 55<sub>16</sub> is stored at address 00009<sub>16</sub>. These two bytes are followed by the least significant byte of the segment base address, 00<sub>16</sub>, at address 0000A<sub>16</sub>, and its most significant byte, A0<sub>16</sub>, at address 0000B<sub>16</sub>. Since the double word is stored in memory starting at starting at address 00008<sub>16</sub>, it is aligned.

# Data Types

- Integer data type
  - Unsigned or signed integer
  - Byte-wide or word-wide integer

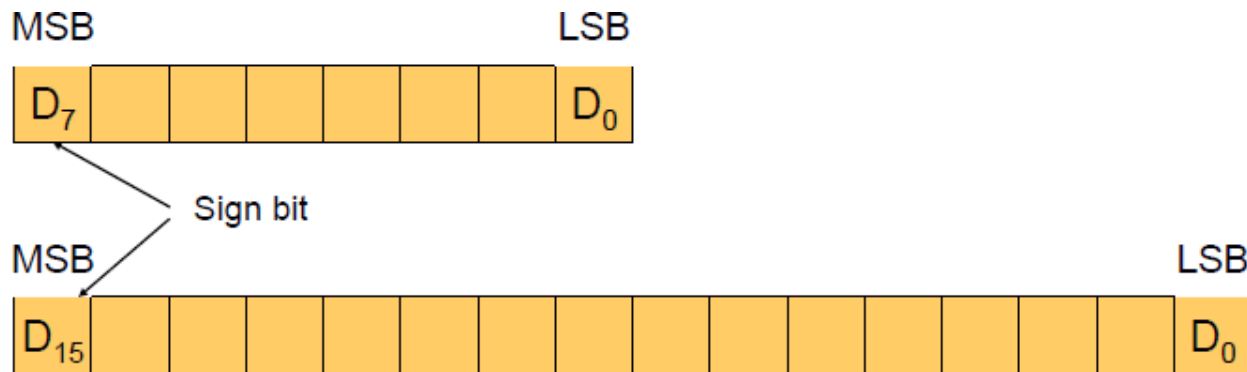
*Byte : 8 bits*  
*Word : 16 bits*  
*Double word : 32 bits*



Unsigned byte and unsigned word integer

# Data Types

- The most significant bit of a signed integer is a sign bit. A zero in this bit position identifies a positive number.
- The range of a signed byte integer is  $+127 \sim -128$ . The range of a signed word integer is  $+32767 \sim -32768$ .
- The 8088 always expresses negative number in 2's complement.



# ASCII Table

- The ASCII ( American Standard Code for Information Interchange) – information expressed here can be directly processed by 8088 microprocessor.

Table shows how numbers, letters, characters coded in ASCII

				b <sub>7</sub>	0	0	0	0	1	1	1	1
				b <sub>6</sub>	0	0	1	1	0	0	1	1
				b <sub>5</sub>	0	1	0	1	0	1	0	1
b <sub>4</sub> b <sub>3</sub> b <sub>2</sub> b <sub>1</sub>			H <sub>1</sub>		0	1	2	3	4	5	6	7
			H <sub>0</sub>									
0 0 0 0	0	NUL	DLE	SP	0	@	P	'	p			
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q			
0 0 1 0	2	STX	DC2	"	2	B	R	b	r			
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s			
0 1 0 0	4	EOT	DC4	\$	4	D	T	d	t			
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u			
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v			
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w			
1 0 0 0	8	BS	CAN	(	8	H	X	h	x			
1 0 0 1	9	HT	EM	)	9	I	Y	i	y			
1 0 1 0	A	LF	SUB	*	:	J	Z	j	z			
1 0 1 1	B	V	ESC	+	;	K	[	k	}			
1 1 0 0	C	FF	FS	,	<	L	\	l				
1 1 0 1	D	CR	GS	-	=	M	]	m	{			
1 1 1 0	E	SO	RS	.	>	N	^	n	~			
1 1 1 1	F	SI	US	/	?	O	-	o	DEL			

# Data Types

## EXAMPLE”

Byte addresses  $01100_{16}$  through  $01104_{16}$  contain the ASCII data 01000001, 01000011, 01001001 and 01001001 respectively. What do the data stand for?

Solution:

Using the ASCII table, the data are converted to ASCII code.

$(01100H) = 01000001_2 = A$

$(01101H) = 01010011_2 = S$

$(01102H) = 01000011_2 = C$

$(01103H) = 01001001_2 = I$

$(01104H) = 01001001_2 = I$



# Instruction Pointer

- The ***instruction pointer (IP)*** identifies the location of the next word of instruction code to be fetched from the current code segment of memory.
- Contains offset of the word of instruction code instead of its actual address.
- This is because both IP and CS are 16 bits in length, 20 bits is require to access the memory.
- The offset in IP is combined with the current value in CS to generate te address of the instruction code. Thus denoted as CS:IP.
- During normal operation, the 8088 fetches instrcutions from the code segment of memory, stores them in its instruction queue and executes them one after the other.

# Data Registers

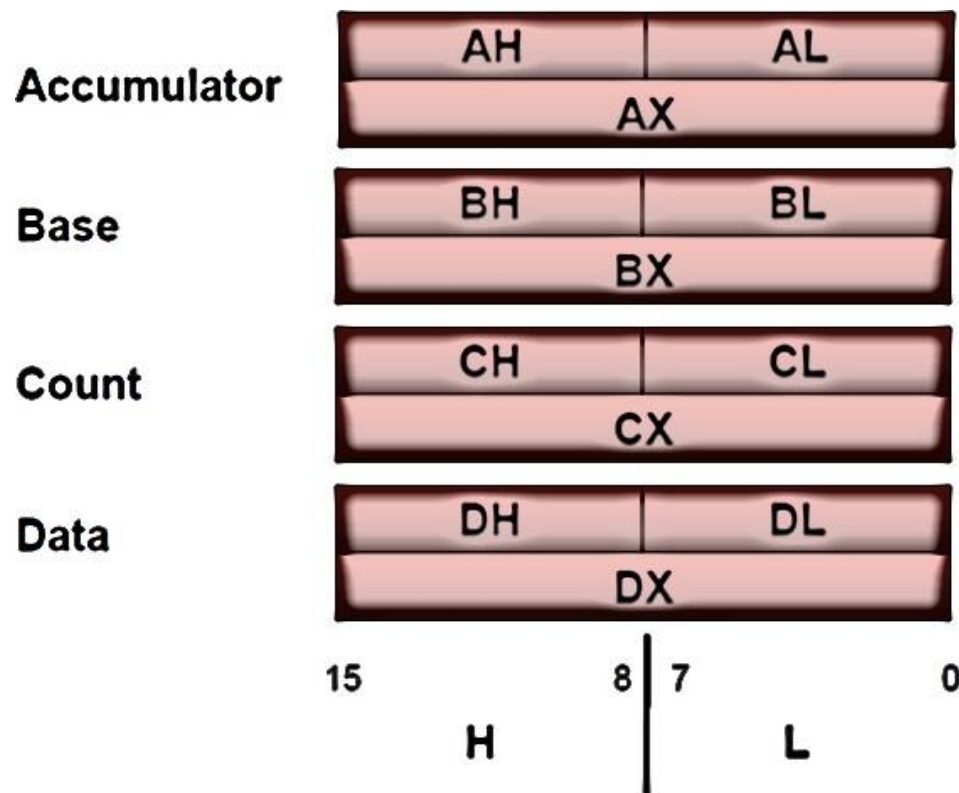
- Data registers are used for temporary storage of frequently used intermediate results.
- The contents of the data registers can be read, loaded or modified through software.
- The four registers are:
  - ❖ Accumulator register, A
  - ❖ Base register, B
  - ❖ Counter register, C
  - ❖ Data register, D



# Data Registers

- Each register can be accessed either as a whole (16 bits) for word data or as 8 bit data for byte wide operation.
- The advantage of storing data in internal registers instead of memory during processing is that they can be accessed much faster.

# Data Registers



General-purpose data registers of 8088 microprocessor

An **X** after the register letter identifies the reference of a register as a word.

When referencing one of these registers on a byte wide basis, with H and L.  
**H** -> high byte (MSB)  
**L** -> Low byte (LSB)

# Multipurpose Registers

- **AX (Accumulator)**

- ☐ Exists as 16-bit register or as either two 8-bit registers (AH and AL)
- ☐ Used for instructions such as multiplication, division and some of the adjustment instructions.

- **BX (Base index)**

- ☐ Can be appear as BH or BL as well.
- ☐ Holds the offset address of a location in the memory systems in all versions of the microprocessor.

- **CX (Count)**

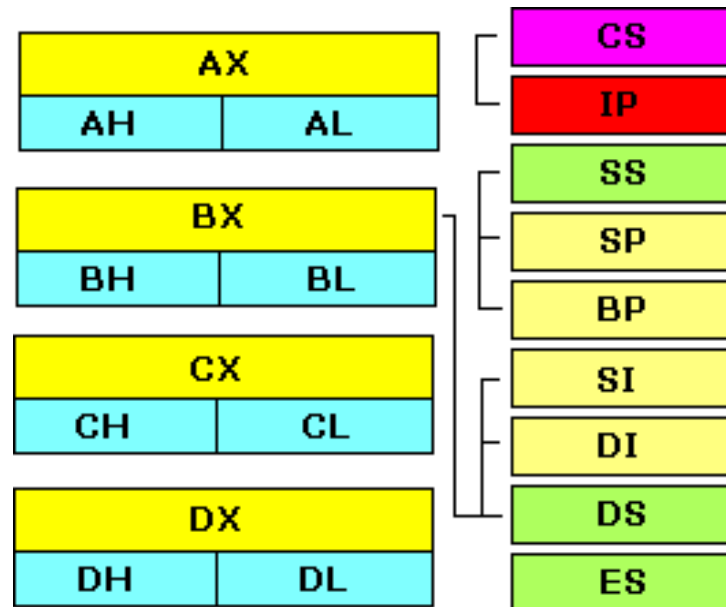
- ☐ Includes CH and CL
- ☐ Holds the count for various instructions including repeated string instructions and shift, rotate and LOOP instructions.

## Pointer & Index Registers

- SP (stack pointer) used to address data in a LIFO (last-in, first-out) stack memory
- BP (base pointer) often used to address an array of data in the stack memory

- The default segments:

- CS:IP
- SS:BP or SP
- DS:SI or DI or BX



# Multipurpose Registers

- **DX (Data)**

- ☐ Holds a part of the result from a multiplication or part of the dividend before a division.

- **BP (Base Pointer)**

- ☐ Points to a memory location in all versions of the microprocessor for memory data transfers.

- **DI (Destination Index)**

- ☐ Addresses string destination data for the string instructions.

- **SI (Source Index)**

- ☐ Addresses source string data for the string instructions.

# Data Registers

Register	Operations
<b>AX</b>	Word multiply, word divide, word I/O
<b>AL</b>	Byte multiply, byte divide, byte I/O, translate, decimal arithmetic
<b>AH</b>	Byte multiply, byte divide
<b>BX</b>	Translate
<b>CX</b>	String operations, loops
<b>CL</b>	Variable shift and rotate
<b>DX</b>	Word multiply, word divide, indirect I/O

Dedicated register functions



# Pointer and Index Register

- The pointer registers and index registers are used to store offset addresses.
- Values held in the index registers are used to reference data relative to the data segment or extra segment.
- The pointer registers are used to store offset addresses of memory location relative to the stack segment register.
- Combining SP with the value in SS ( SS:SP) results in a 20 bit address that points to the top of the stack (TOS).
- BP is used to access data within the stack segment of memory. It is commonly used to reference subroutine parameters.



# Segment Registers

## ■ CS Register

- Starting address of the memory segment that contains instructions of the program
- The contents are combined with the values in IP to generate a physical address

## ■ DS Register

- Starting location of the data segment which is where data can be stored, and operands are fetched from
- Combined with values in SI or DI

# Segment Registers

- SS register
  - Starting address of the stack segment, a segment where the values of IP, status flags and other registers are pushed whenever an interrupt or subroutine calls
  - After the completion of the interrupt service routine or subroutine, the original system status is restored from the stack using the pop and return instructions
  - Combined with the value in SP to identify the next location a word is to be pushed or popped



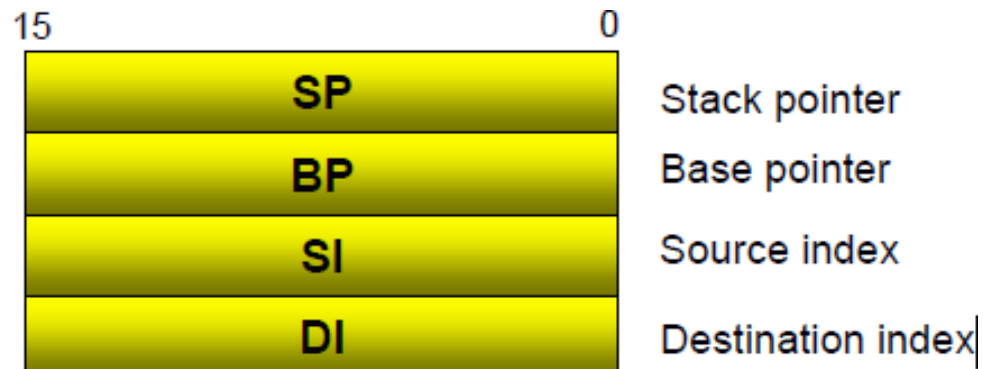
# Segment Registers

## ■ ES Register

- Starting address of the extra segment, which is usually used for data storage
- The contents may be used with the contents of DI to specify a destination address

# Pointer and Index Register

- The index register are used to hold offset addresses for instructions that access data in the data segment.
- The source index register (SI) is used for a source operand and the destination index (DI) is used for a destination operand.
- The four registers must always be used for 16 bit operations.



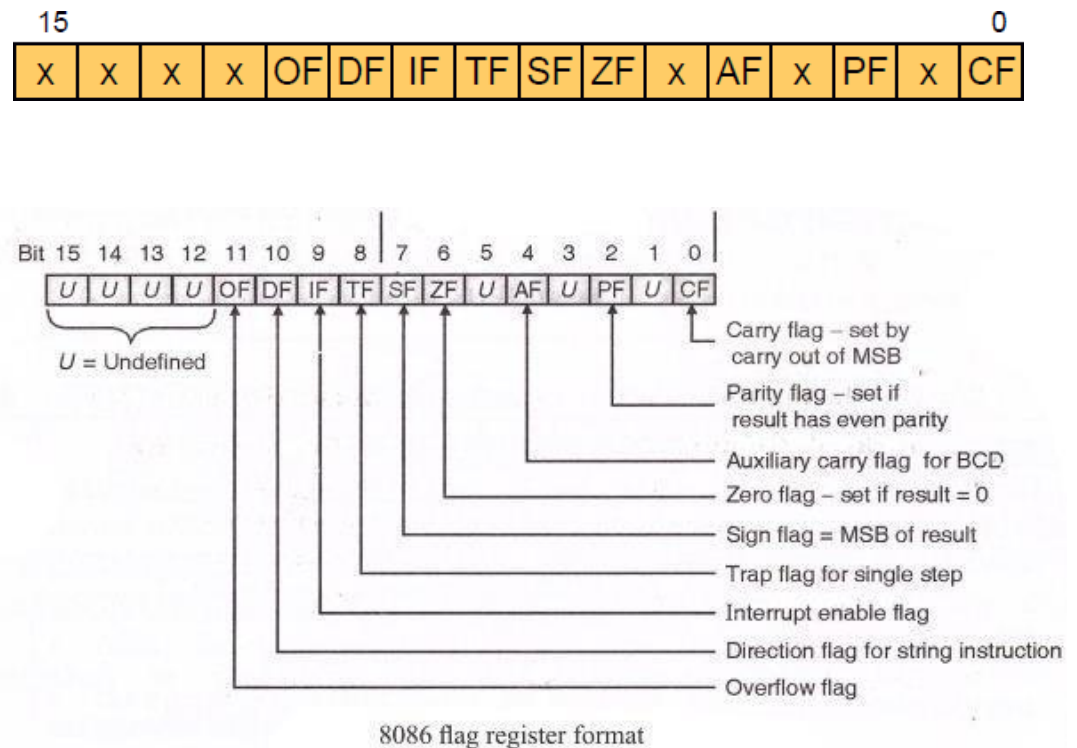


# Status Register

- The status register, also called the *flags register*, indicate conditions that are produced as the result of executing an instruction.
- Only nine bits of the register are implemented. Six of these bits represent *status flags* and the other three bits represent **control flags**.
- The 8088 provides instructions within its instruction set that are able to use these flags to alter the sequence in which the program is executed.

# Status Register

- Status and control bits maintained in the flags register
- Generally Set and Tested individually
- 9 1 bit flags in 8086; 7 are unused



# Status Register

- Status flags indicate current processor status.

<b>CF</b>	Carry Flag	Arithmetic Carry/Borrow
<b>OF</b>	Overflow Flag	Arithmetic Overflow
<b>ZF</b>	Zero Flag	Zero Result; Equal Compare
<b>SF</b>	Sign Flag	Negative Result; Non-Equal Compare
<b>PF</b>	Parity Flag	Even Number of "1" bits
<b>AF</b>	Auxiliary Carry	Used with BCD Arithmetic



# Status Register

- Control flags influence the 8086 during execution phase.

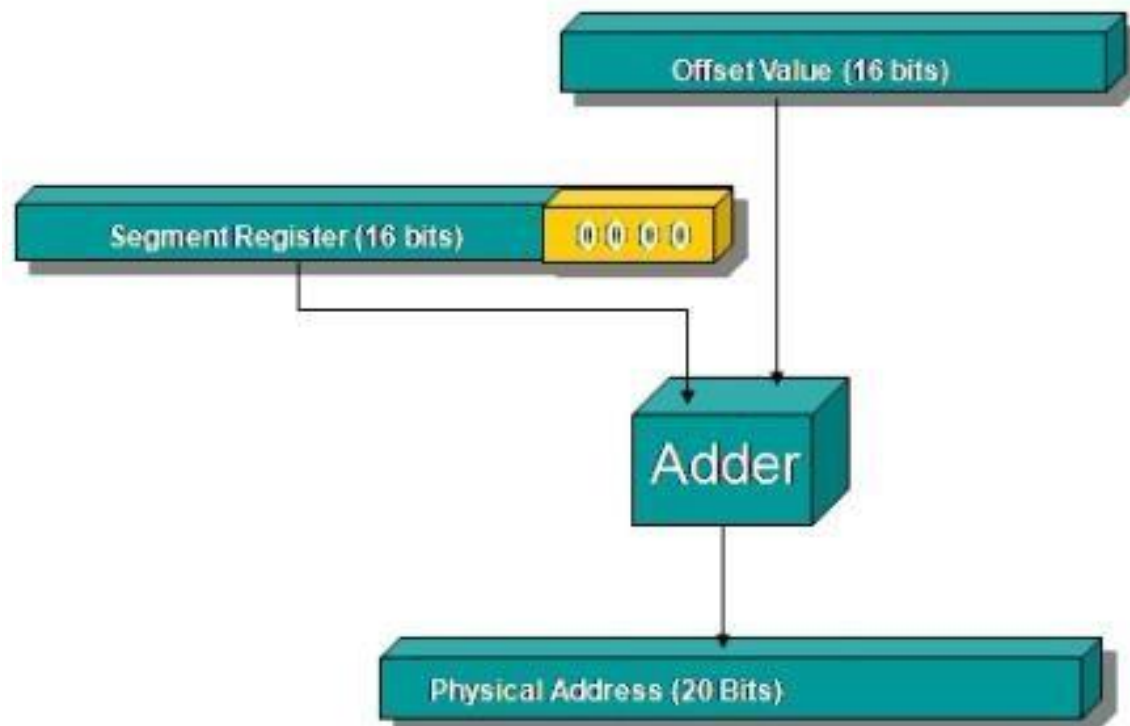
**DF**      Direction Flag      Auto-Increment/Decrement  
used for “string operations”

**IF**      Interrupt Flag      Enables Interrupts  
allows “fetch-execute” to be interrupted

**TF**      Trap Flag      Allows Single-Step  
for debugging; causes interrupt after each op

# Segmentation

- Memory Address Generation: The BIU has a dedicated adder for determining physical memory addresses.



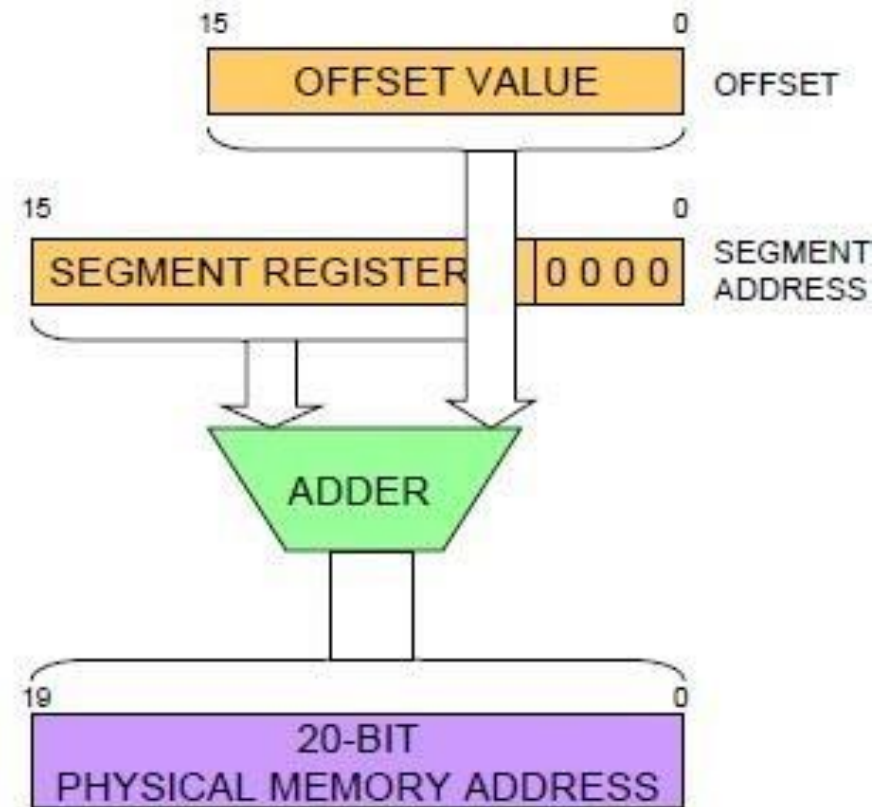
# Memory Address Generation

Offset value denote the number of address locations added to the based address in order to get a specific absolute address

Segment register stores the starting addresses of a segment. To get the exact location of data or instruction within a segment, an offset value (or displacement) is required

Adder perform addition of number

Physical address- real address or binary address

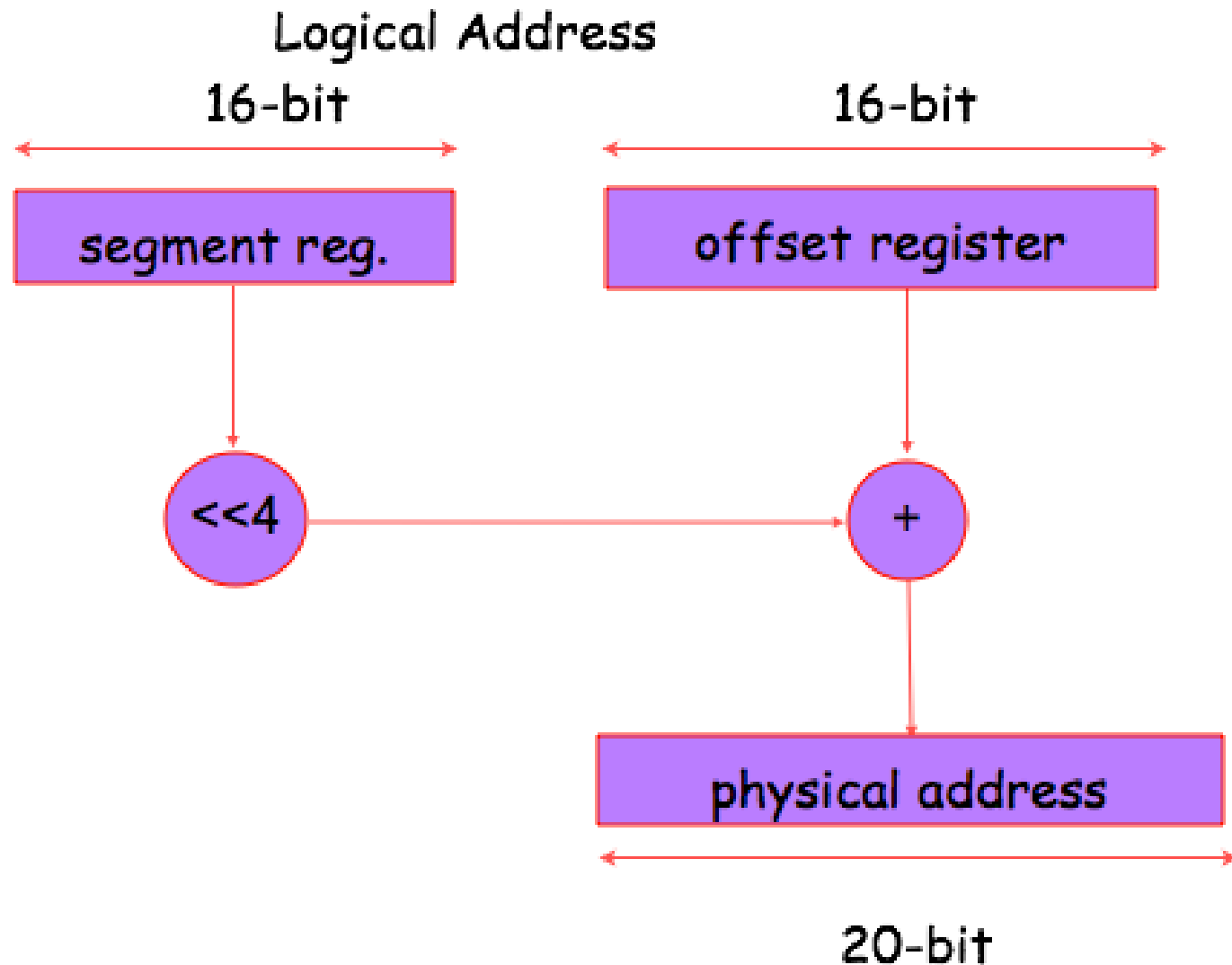


# Memory Address Generation

- The base value can be from
  - CS (code segment), DS (data segment), SS (stack segment) or ES (extra segment)
- The point to note is that the beginning segment address is not arbitrary - *it must begin at an address divisible by 16*, Another way of saying this is that the low-order hex digit must be 0.

Type of memory reference	Default segment	Alternate segment	Offset (logical address)
Instruction fetch	CS	None	IP
Stack operation	SS	None	SP
General data	DS	CS, ES, SS	Effective address
String source	DS	CS, ES, SS	SI
String destination	ES	None	DI
BX used as pointer	DS	CS, ES, SS	Effective address
BP used as pointer	SS	CS, ES, DS	Effective address

# Physical & Logical Address



# Physical & Logical Address

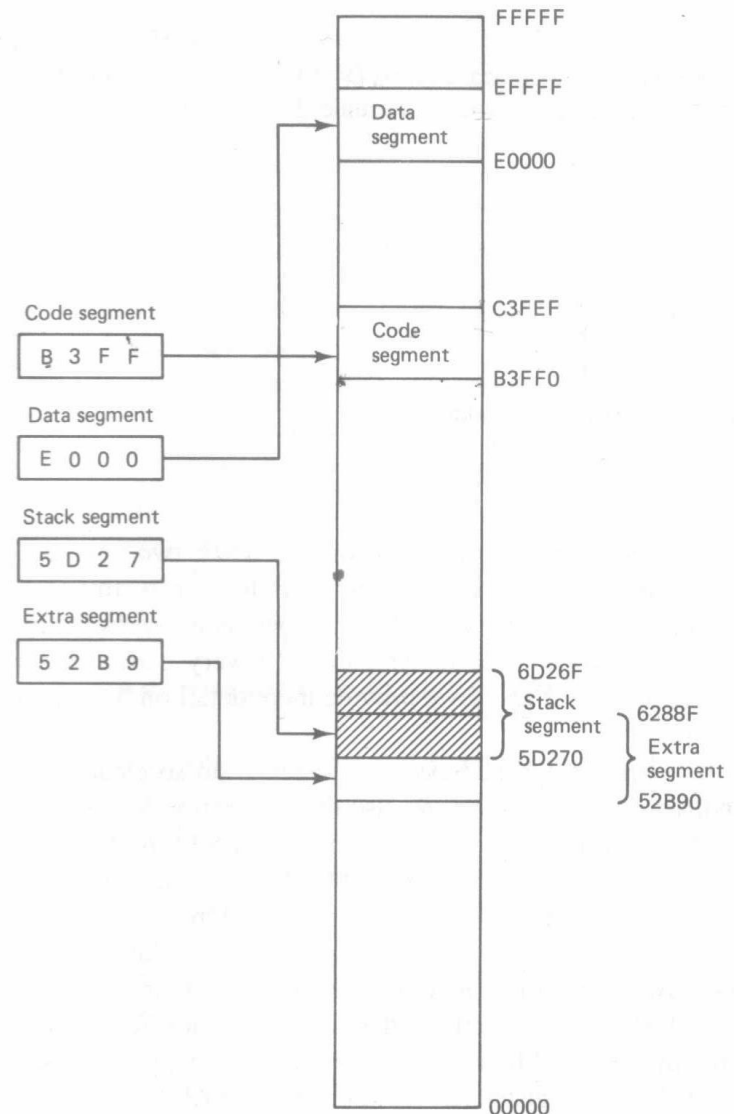
## ■ **Example:**

Calculate the physical address corresponding to logical address D470H in the **extra segment**. Repeat for logical address 290AH in the **stack segment**. Assume the segment definitions shown in Figure A above.

## **Solution**

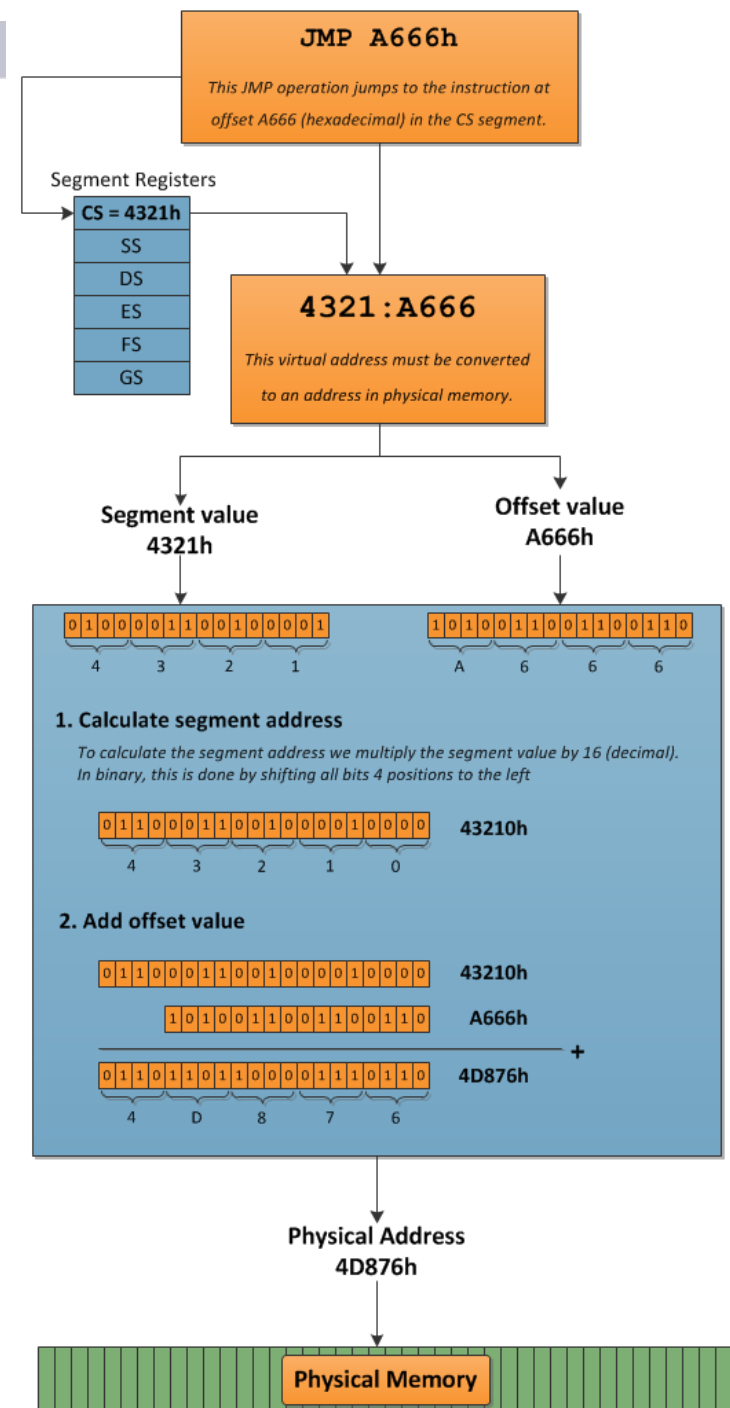
For the extra segment  
52B90H (base address)  
+ D470H (offset)  
60000H

and for the stack segment  
5D270H (base address)  
+ 290AH (offset)  
5FB7AH



# Physical & Logical Address

- Logical Address (offset) = 4321H: A666H
- Physical Address = 4D876H
- Physical Address is also known as Absolute Address.
- Calculation:
- $CS \times 10H + \text{Offset} = 4321H \times 10H + A666H = 4D876H$ .





# Advantages of Segmented Memory

- Program op-codes will be fetched from the code segment, while program data variables will be stored in the data and extra segments. Stack operations use registers BP or SP and the stack segment.
- Program can work on several different sets of data. This is done by reloading-register DS to point to the new data.
- Reference logical addresses can be loaded and **run anywhere** in memory. This is because the logical addresses always range from 0000 to FFFFH, independent of the code segment base.



# Advantages of Segmented Memory

- Consider a *multitasking* environment in which the 8086 is doing several different jobs at once.
- An inactive program can be temporarily saved on a magnetic disk and a new program brought in to take its place - without concern for the physical location of this new program.
- Such programs are said to be **relocatable**, meaning that they will run at any location in memory.
- The requirements for writing relocatable programs are that no references be made to physical addresses, and no changes to the segment registers are allowed.

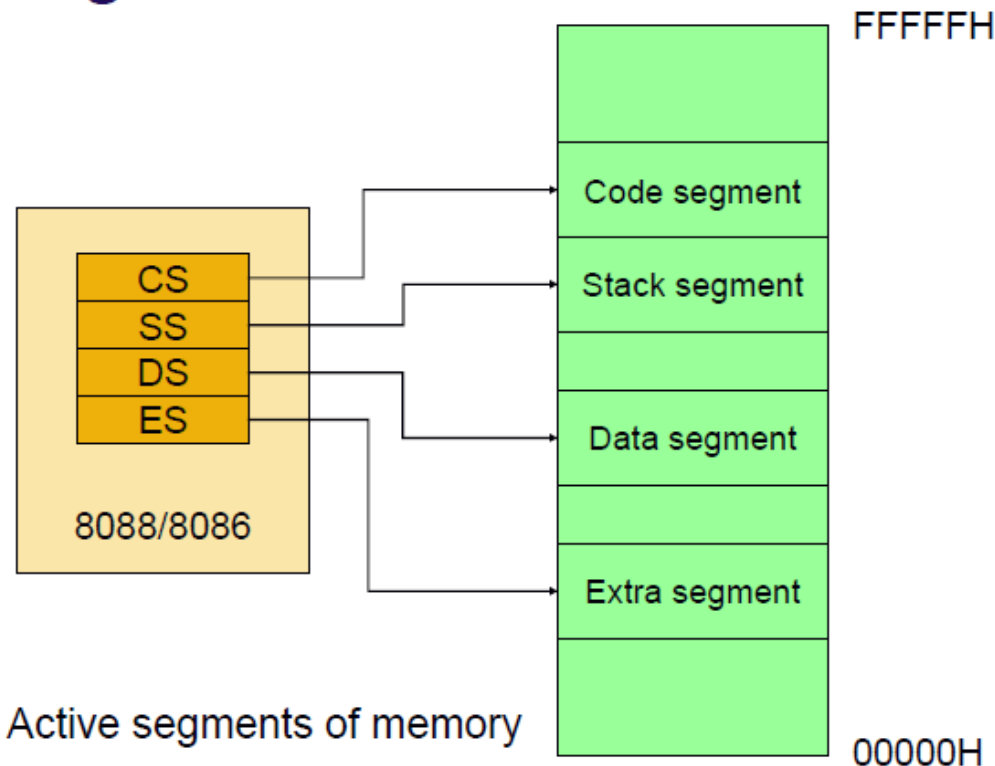


# Segment Registers and Memory Segmentation

- 8088 has 1 Mbyte of memory but not all are active at one time.
- Is partitioned into 64Kbyte (65536) segments
- Each segment represents independently addressable units of memory of 64K consecutive byte storage locations
- Each segment is assigned a base address that identifies its starting point.
- Only 4 of these 64Kbyte segments are active at a time
  - ❖ Code – stores program or instruction codes
  - ❖ Data – stores data for the program
  - ❖ Stack - store interrupt and subroutine return addresses
  - ❖ Extra – additional data segment.

# Segment Registers and Memory Segmentation

- The addresses of the active segments are stored in the four internal segment registers: CS, SS, DS, ES.



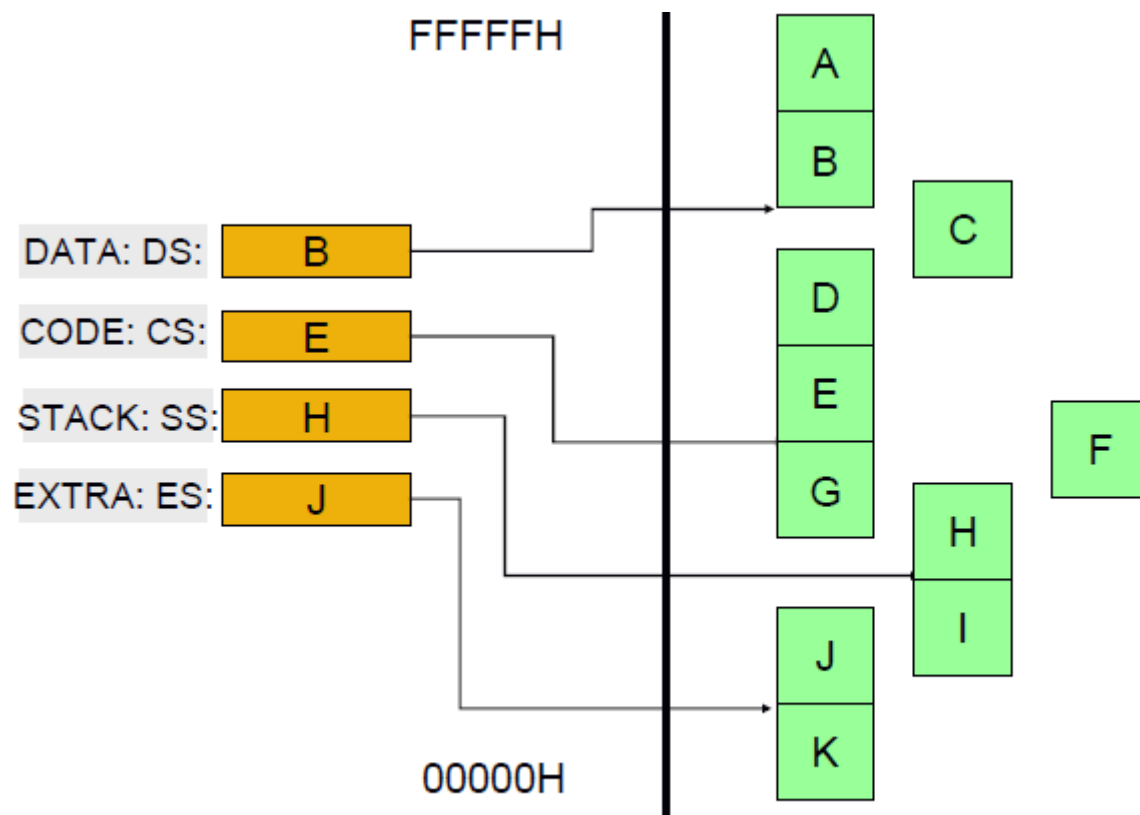
Each of these registers contains a 16 bit base address that points to the lowest addressed byte of the segment in memory.



# Segment Registers and Memory Segmentation

- Four segments give a maximum of 256Kbytes of active memory.
  - ❖ Code segment -64K
  - ❖ Stack - 64K
  - ❖ Data Storage – 128K
- The base address of a segment must reside on a 16-byte address boundary.
- User accessible segments can be set up to be contiguous, adjacent, disjoint or overlapping

# Segment Registers and Memory Segmentation

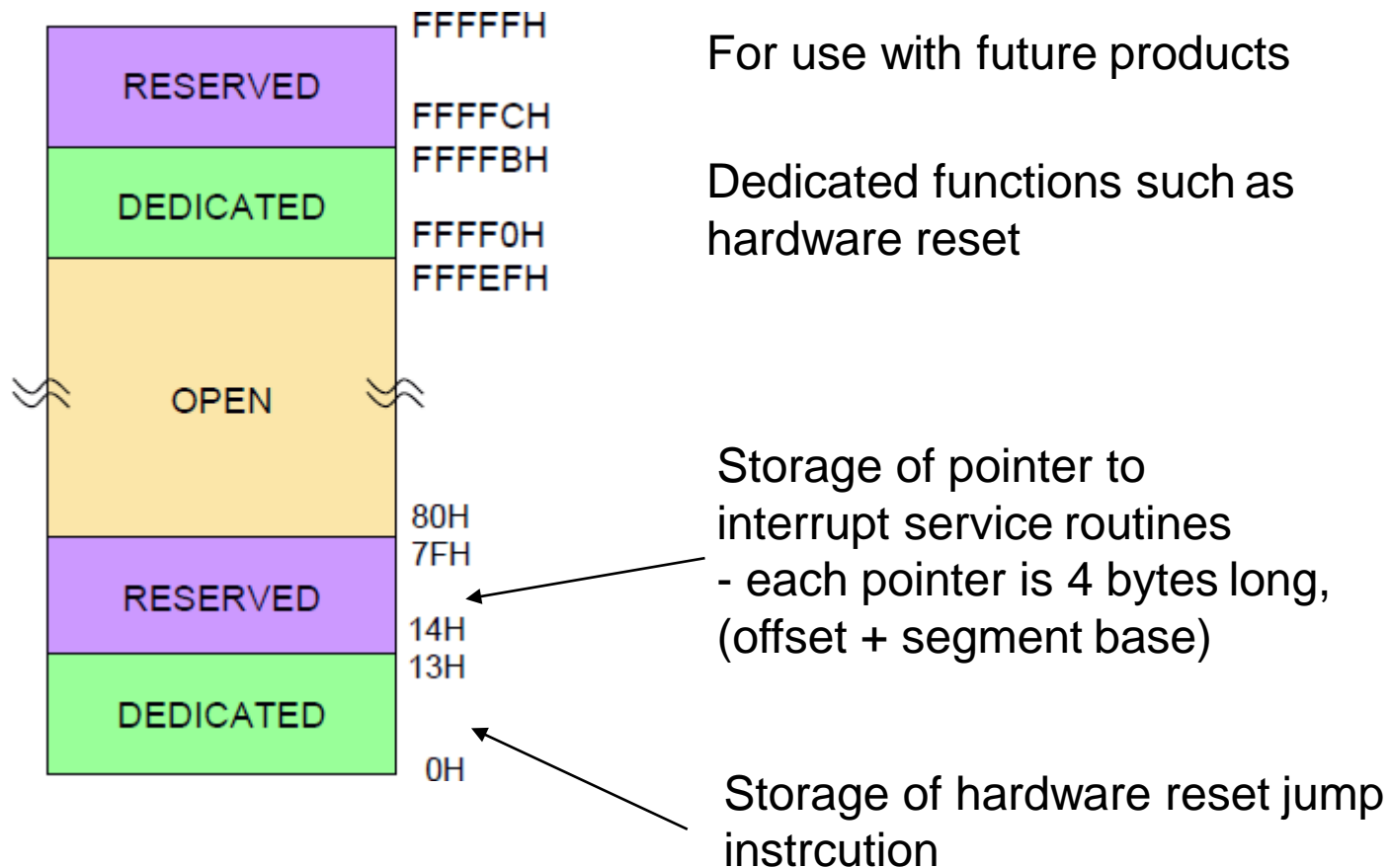


Contiguous, adjacent, disjointed, and overlapping segments

# Dedicated, Reserved and General-Used Memory

- The ***dedicated memory*** ( $00000_{16} \sim 00013_{16}$ ) are used for storage of the pointers to 8088's internal interrupt service routines and exceptions.
- The ***reserved memory*** ( $00014_{16} \sim 0007F_{16}$ ) are used for storage of the pointers to user-defined interrupts.
- The 128- byte dedicated and reserved memory can contain 32 interrupt pointers.
- The ***general-use memory*** ( $00080_{16} \sim FFFEF_{16}$ ) store data or instructions of the program.
- The dedicated memory ( $FFFE0_{16} \sim FFFEB_{16}$ ) are used for hardware reset jump instruction.

# Dedicated, Reserved and General-Used Memory



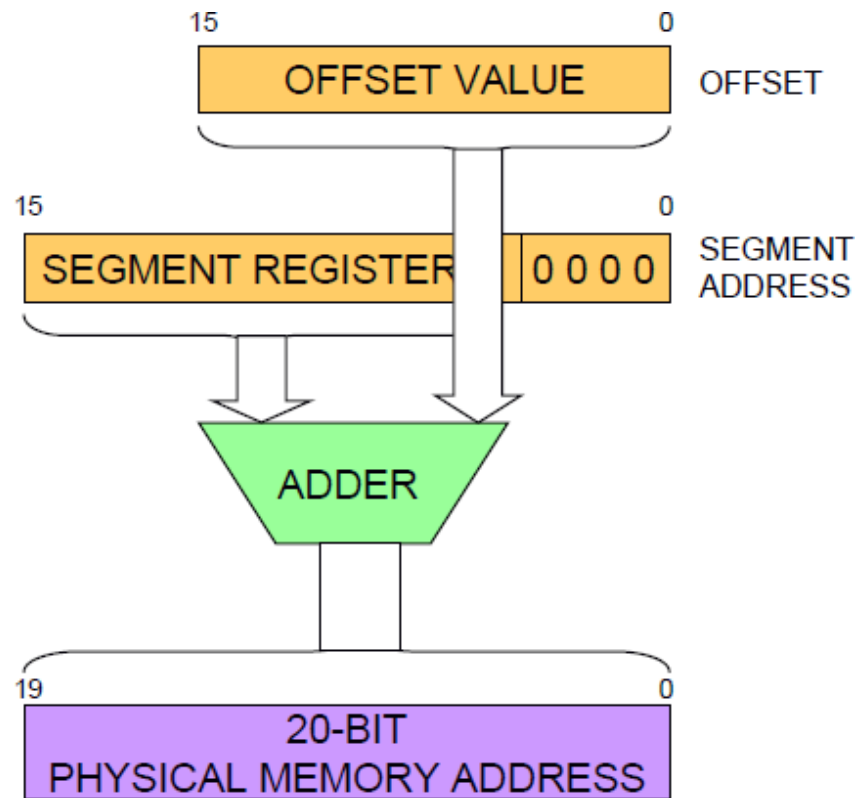


# Generating a Memory Address

- A logical address in the 8088 microprocessor system is described by a segment base and an offset which is 16 bits.
- The physical addresses that are used to access memory are **20 bits** in length.
- The generation of the physical address involves combining a 16 bit offset value that is located in the instruction pointer, a base pointer, an index register, or a pointer register and a 16 bit segment base value that is located in one of the segment register.

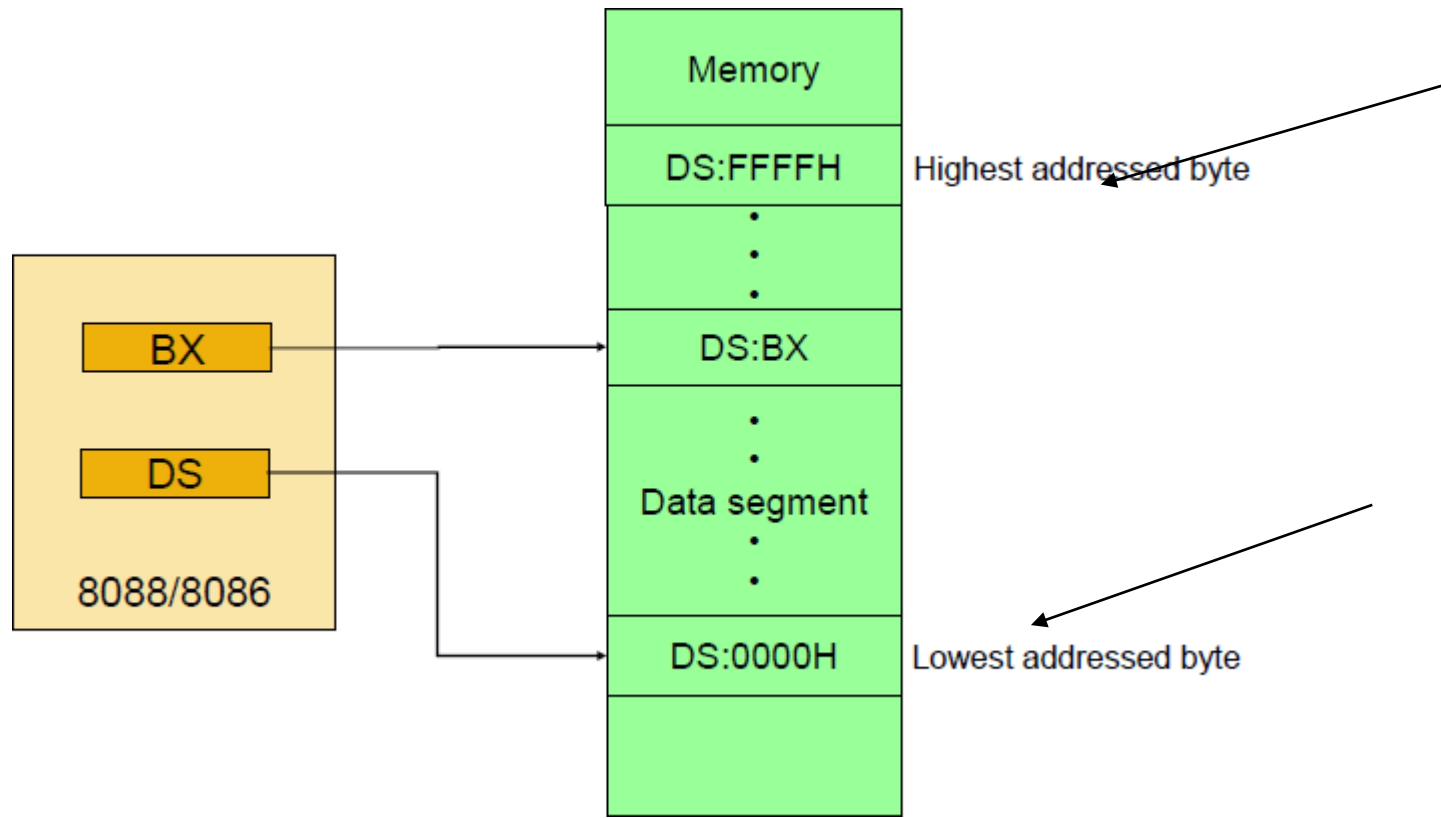


# Generating a Memory Address



Generating a physical address

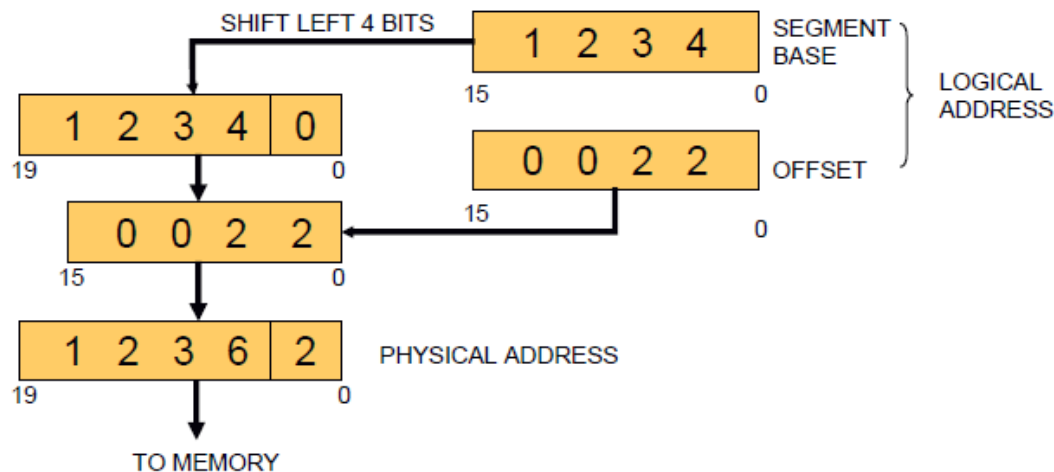
# Generating a Memory Address



# Generating a Memory Address

- A segment base value of  $1234_{16}$  and offset value of  $0022_{16}$ .

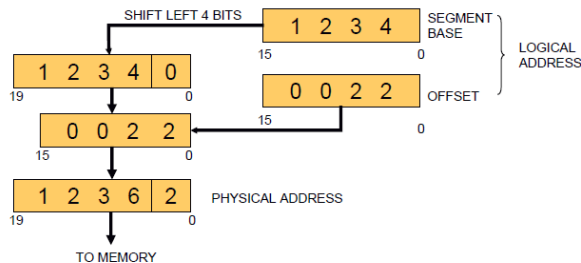
EXAMPLE



# Generating a Memory Address

- A segment base value of  $1234_{16}$  and offset value of  $0022_{16}$ .

## EXAMPLE



1. First express base value in binary form.

$$1234_{16} = 0001001000110100_2$$

2. Shifting left with four positions and filling with zeros results in the segment address.

$$00010010001101000000_2 = 12340_{16}$$

3. The offset address in binary form is

$$0022_{16} = 0000000000100010_2$$

4. Adding the segment address and the offset gives

$$00010010001101000000_2 +$$

$$0000000000100010_2 =$$

$$00010010001101100010_2 = 12362_{16} = 12362H$$