

CPE 408330

Assembly Language and Microprocessors

Chapter 2: Software Architecture of the 8088 and 8086 Microcomputers

[Computer Engineering Department,
Hashemite University]

Lecture Outline

- ▶ 2.1 Microarchitecture of the 8088/8086 Microprocessor
- ▶ 2.2 Software Model of the 8088/8086 Microprocessor
- ▶ 2.3 Memory Address Space and Data Organization
- ▶ 2.4 Data Types
- ▶ 2.5 Segment Registers and Memory Segmentation
- ▶ 2.6 Dedicated, Reserved, and General-Used Memory
- ▶ 2.7 Instruction Pointer
- ▶ 2.8 Data Registers

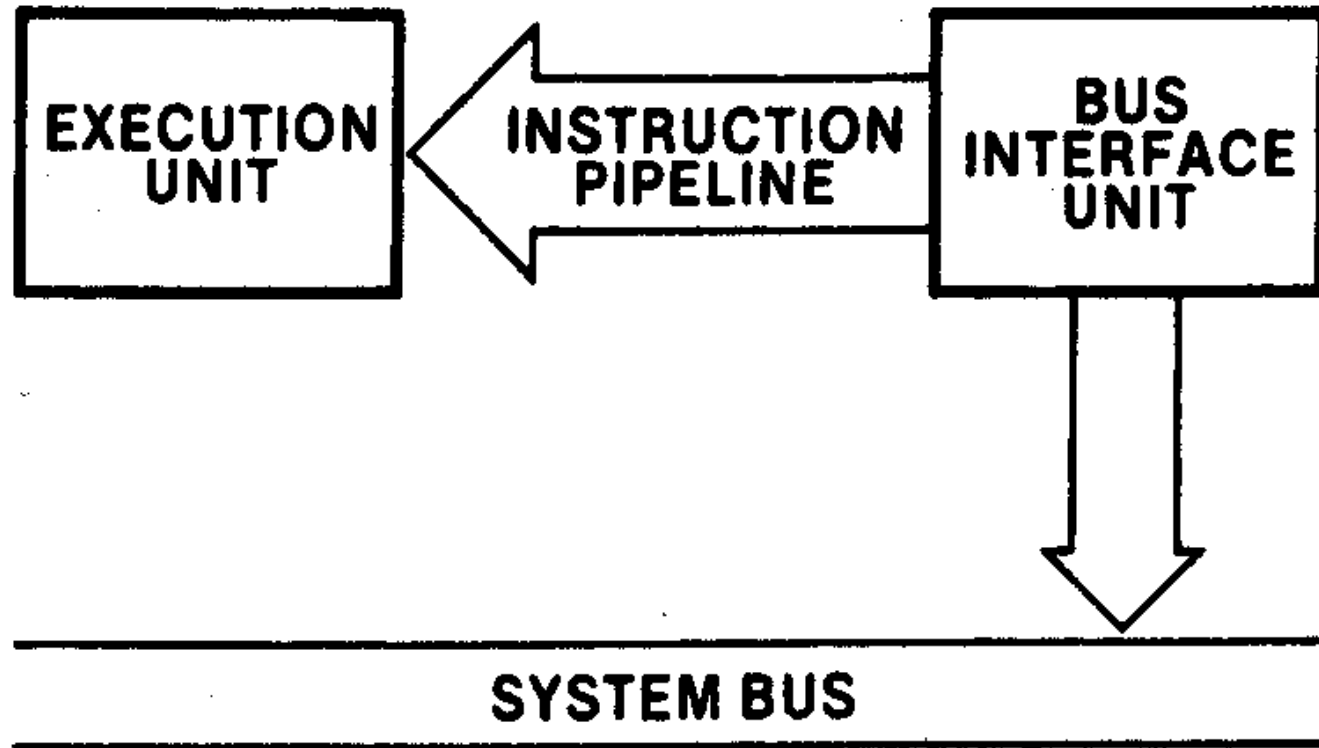
Lecture Outline

- ▶ 2.9 Pointer and Index Register
- ▶ 2.10 Status Register
- ▶ 2.11 Generating a Memory Address
- ▶ 2.12 The Stack
- ▶ 2.13 Input/output Address Space

2.1 Microarchitecture of the 8088/8086 Microprocessor

- ▶ 8088/8086 both employ *parallel processing*
- ▶ 8088/8086 contain two processing unit – the *bus interface unit (BIU) and execution unit (EU)*
- ▶ The bus interface unit is the path that 8088/8086 connects to *external devices*.
- ▶ The system bus includes an 8-bit bidirectional *data* bus for 8088 (16 bits for the 8086), a 20-bit *address* bus, and the signal needed to *control* transfers over the bus.

2.1 Microarchitecture of the 8088/8086 Microprocessor

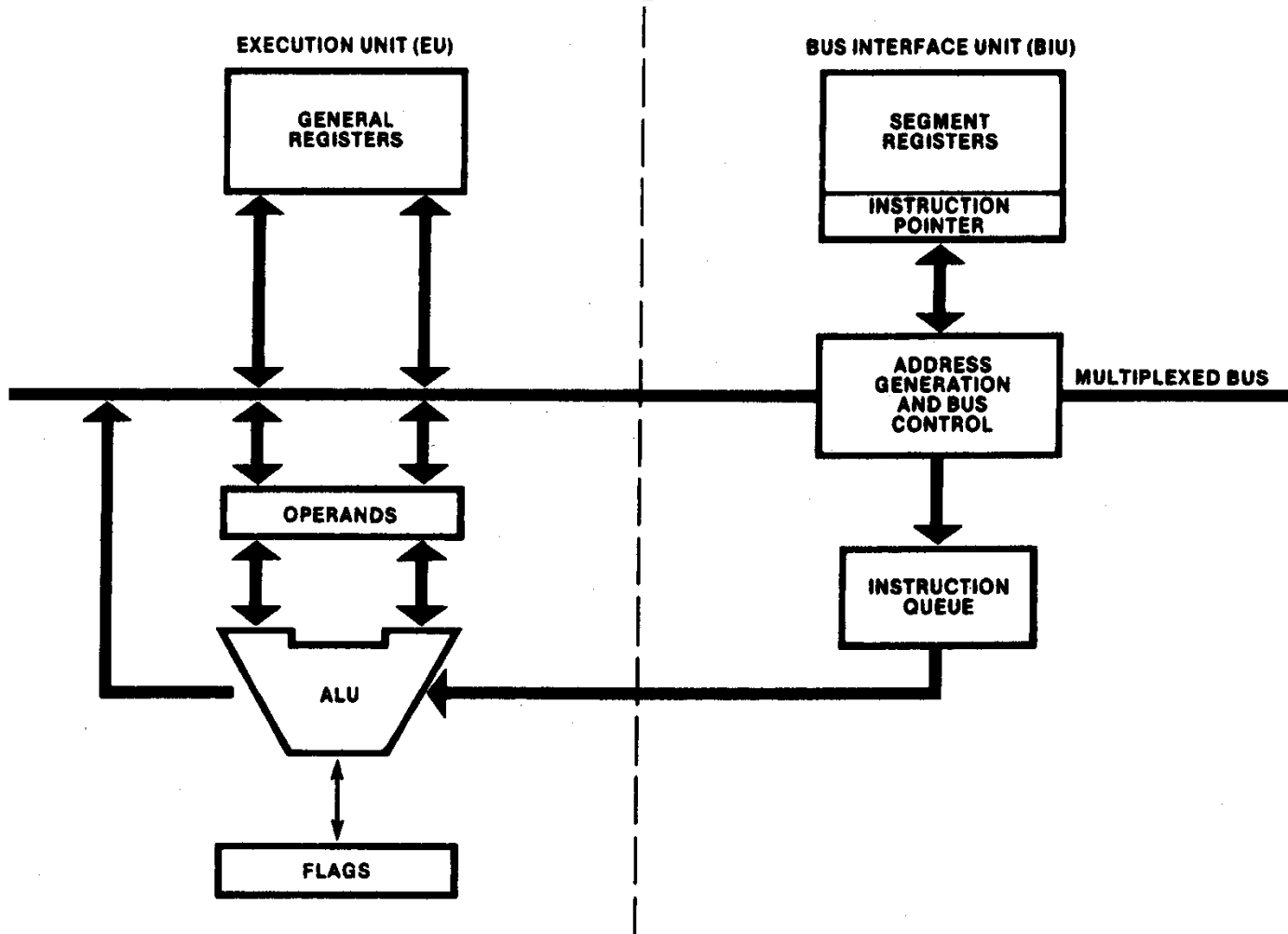


Pipeline architecture of the 8086/8088 microprocessors

2.1 Microarchitecture of the 8088/8086 Microprocessor

- ▶ BIU is responsible for: Instruction fetching, memory reading/writing and inputting/outputting data for peripherals.
- ▶ **Components in BIU**
 - Segment register
 - The instruction pointer
 - Address generation adder
 - Bus control logic
 - Instruction queue
- ▶ **Components in EU**
 - Arithmetic logic unit, ALU
 - Status and control flags
 - General-purpose registers
 - Temporary-operand registers

2.1 Microarchitecture of the 8088/8086 Microprocessor

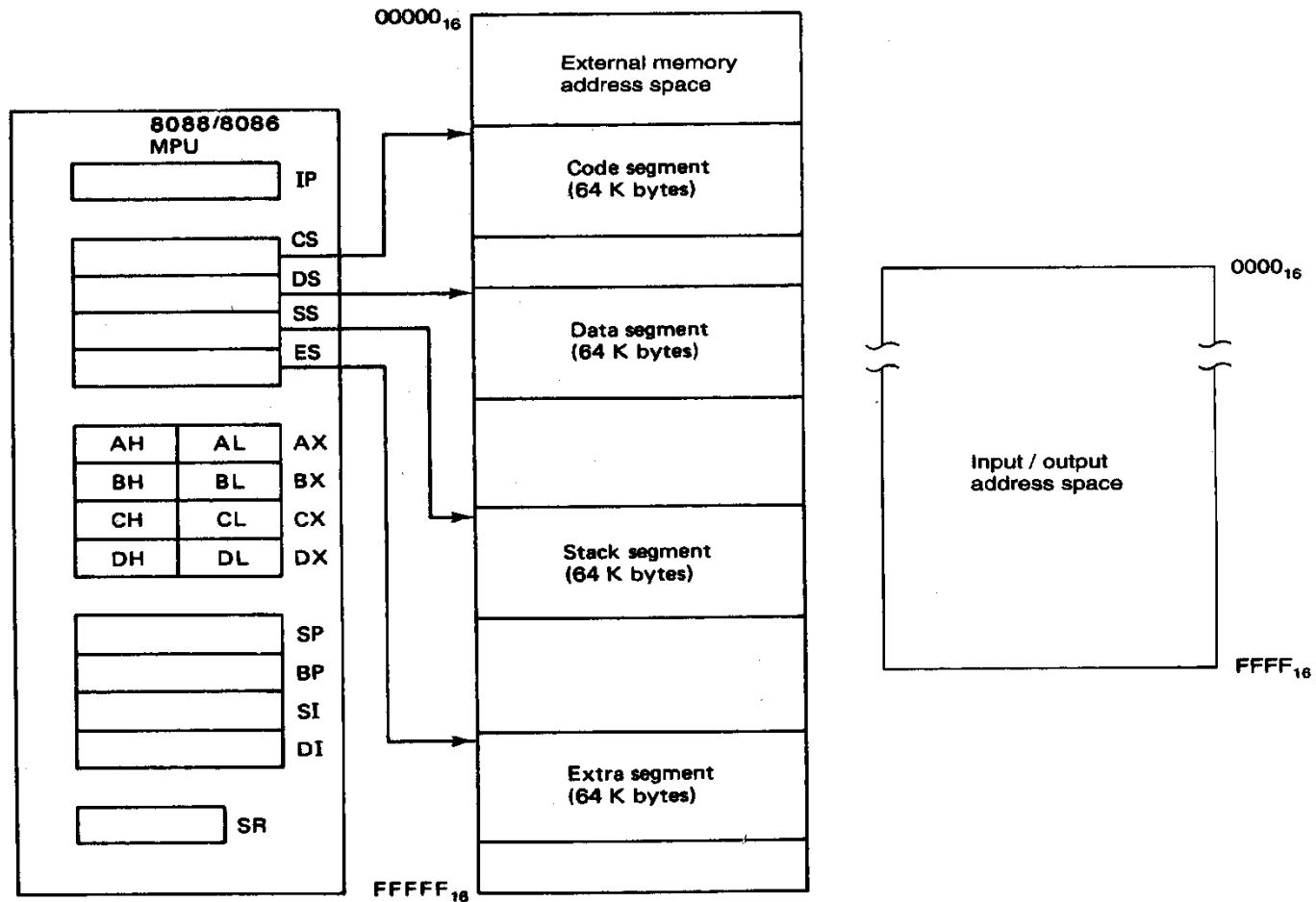


EU and BIU of the 8086/8088 microprocessors

2.2 Software Model of the 8088/8086 Microprocessor

- ▶ Software model describes: available registers, memory address space and I/O address space.
- ▶ 8088 microprocessor includes 13 **16-bit** internal registers.
 - The instruction pointer, IP
 - Four data registers, AX, BX, CX, DX
 - Two pointer register, BP, SP
 - Two index register, SI, DI
 - Four segment registers, CS, DS, SS, ES
- ▶ The status register, SR, with **nine** of its bits implemented for status and control flags.
- ▶ The memory address space is 1 Mbytes and the I/O address space is 64 Kbytes in length.

2.2 Software Model of the 8088/8086 Microprocessor



Software model of the 8088/8086 microprocessors

2.3 Memory Address Space and Data Organization

- ▶ The 8088 microcomputer supports 1 Mbytes of external memory.
- ▶ The memory of an 8088-based microcomputer is organized as 8-bit bytes, not as 16-bit words.

Memory address
space of the
8088/8086
Microcomputer

FFFFF
FFFFE
FFFFD
FFFFC
5
4
3
2
1
0

2.3 Memory Address Space and Data Organization

- ▶ The 8088 can access any two **consecutive** bytes as **word** of data.
- ▶ **Lower** address byte and **higher** address byte.
- ▶ The two bytes represent the word

Address

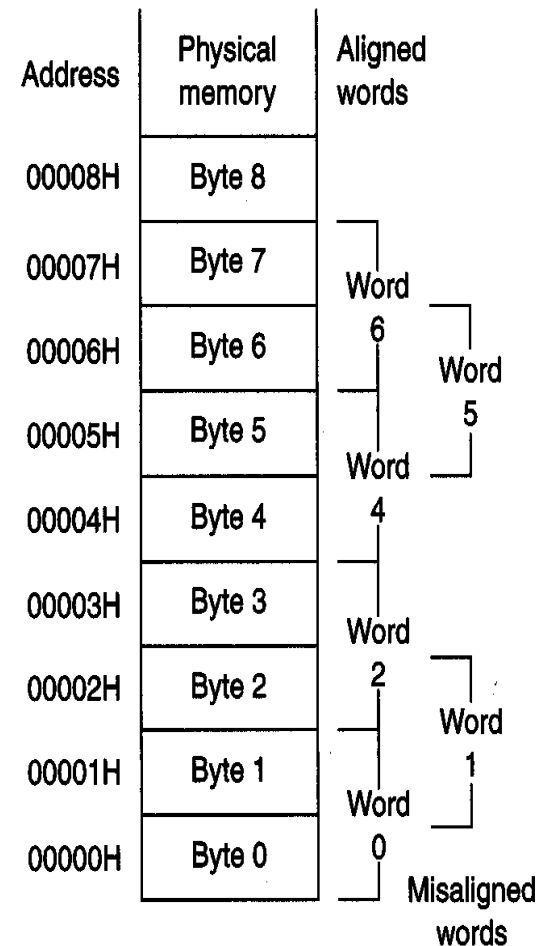
...
00725
00724
...

Memory

...
0101 0101
0000 0010 = 5502₁₆
...

2.3 Memory Address Space and Data Organization

- ▶ **Even**– or **odd-addressed** word
If the least significant bit of the address is 0, the word is said to be held at an even-addressed boundary.
- ▶ **Aligned** word (even-address) or **misaligned** word (odd-address).
- ▶ Words 0, 2, 4 & 6: aligned
- ▶ Words 1 & 5: misaligned



2.3 Memory Address Space and Data Organization

▶ EXAMPLE

What is the data word shown below? Express the result in hexadecimal form. Is it stored at an even- or odd addressed word boundary? Is it an aligned or misaligned word of data?

<u>Address</u>	<u>Memory</u>
0072C	1111 1101
0072B	1010 1010

▶ Solution:

$$11111101_2 = \text{FD}_{16} = \text{FD}_H$$

$$10101010_2 = \text{AA}_{16} = \text{AA}_H$$

▶ Together the two bytes give the word

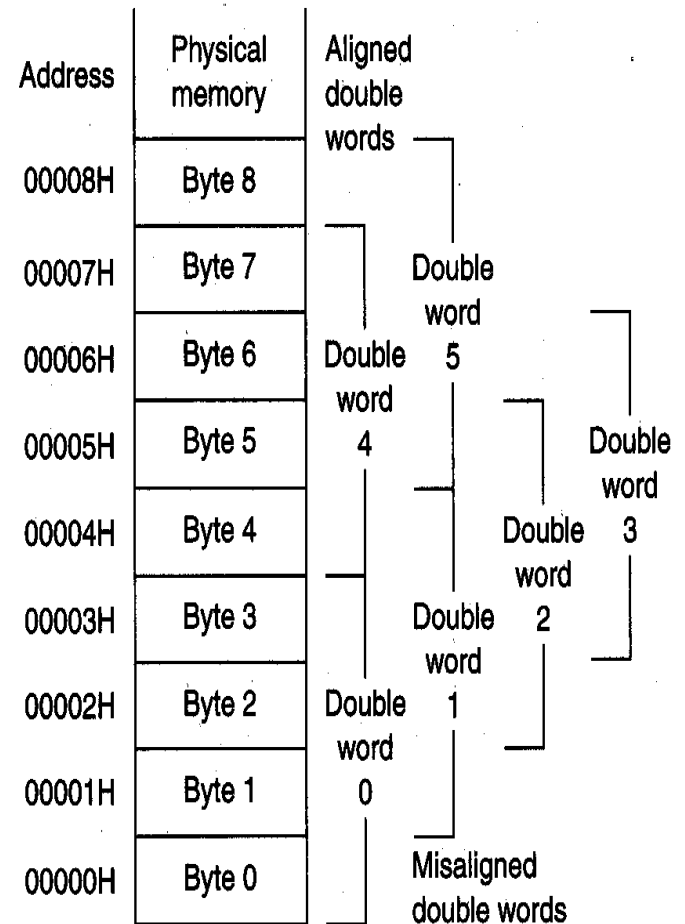
$$1111110110101010_2 = \text{FDAA}_{16} = \text{FDAA}_H$$

Expressing the address of the least significant byte in binary form gives $0072B_H = 0072B_{16} = 00000000011100101011_2$

- ▶ $\text{LSB} = 1 \rightarrow$ the word is stored at odd-address boundary in memory.
- ▶ Therefore, it is misaligned word of data.

2.3 Memory Address Space and Data Organization

- ▶ A *double word* corresponds to **four** consecutive bytes of data stored in memory.
- ▶ Aligned double word is located at addresses of **multiples of 4**.
- ▶ Word 0 & 4: aligned only



2.3 Memory Address Space and Data Organization

- ▶ A **pointer** is a double word. 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 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

Example: Segment base address = $3B4C_{16} = 0011101101001100_2$

- ▶ Offset value = $0065_{16} = 00000000001100101_2$

2.3 Memory Address Space and Data Organization

▶ EXAMPLE

How should the pointer with **content in** segment base address equal to $A000_{16}$ and **content in** offset address equals $55FF_{16}$ be stored at an even-address boundary starting at 00008_{16} ? Is the double word aligned or misaligned?

▶ Solution:

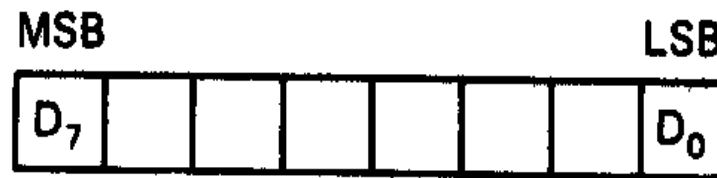
Storage of the two-word pointer requires four consecutive byte locations in memory, starting at address 00008_{16} . The least significant byte of the offset is stored at address 00008_{16} and is shown as FF_{16} in the previous figure. The most significant byte of the offset, 55_{16} , is stored at address 00009_{16} . These two bytes are followed by the least significant byte of the segment base address, 00_{16} , at address $0000A_{16}$, and its most significant byte, $A0_{16}$, at address $0000B_{16}$. Since the double word is stored in memory starting at address 00008_{16} , it is aligned.

Previous Example Solution

Address	Memory (hexadecimal)
$0000B_{16}$	A0
$0000A_{16}$	00
00009_{16}	55
00008_{16}	FF

2.4 Data Types

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



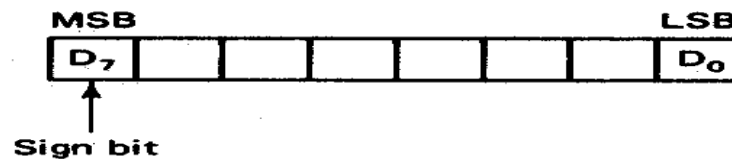
(a)



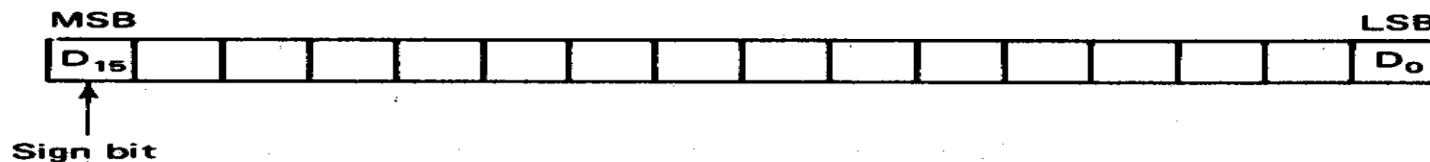
Unsigned byte and unsigned word integer

2.4 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** numbers in **2's complement**.



(a)



(b)

signed byte and signed word integer

2.4 Data Types

▶ EXAMPLE

A signed word integer equals FEFF_{16} . What decimal number does it represent?

▶ Solution:

$$\text{FEFF}_{16} = 1111111011111111_2$$

- The most significant bit is 1, the number is negative and is in 2's complement form.
- Converting to its binary equivalent by subtracting 1 from the least significant bit and then complement all bits give

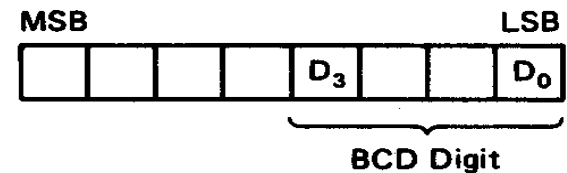
$$\begin{aligned}\text{FEFF}_{16} &= -0000000100000001_2 \\ &= -257\end{aligned}$$

2.4 Data Types

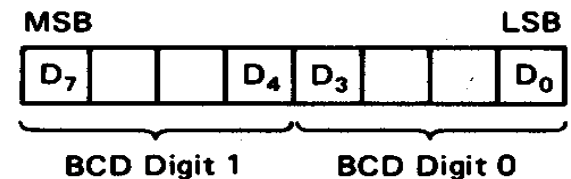
- ▶ The 8088 can also process data that is coded as *binary-coded decimal (BCD) numbers*.
- ▶ BCD data can be stored in **unpacked** (upper 4 bits = 0) or **packed** forms.

Decimal	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

(a)



(b)



(c)

(a) BCD numbers (b) Unpacked BCD digit (c) Packed BCD digit

2.4 Data Types

▶ EXAMPLE

The packed BCD data stored at byte address 01000_{16} equals 10010001_2 . What is the two digit decimal number?

▶ Solution:

Writing the value 10010001_2 as separate BCD digits gives

$$10010001_2 = 1001_{\text{BCD}}0001_{\text{BCD}} = 91_{10}$$

2.4 Data Types

- ▶ The ASCII (American Standard Code for Information Interchange) digit
- ▶ The 8088 can process ASCII characters too.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(72	48	H	104	68	h
9	09	Horizontal tab	41	29)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

2.4 Data Types

▶ EXAMPLE

Byte addresses 01100_{16} through 01104_{16} contain the ASCII data 01000001, 01010011, 01000011, 01001001, and 01001001, respectively. What do the data stand for?

▶ Solution:

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

$(01100H) = 01000001_2 = A$

$(01101H) = 01010011_2 = S$

$(01102H) = 01000011_2 = C$

$(01103H) = 01001001_2 = I$

$(01104H) = 01001001_2 = I$

2.5 Segment Registers and Memory Segmentation

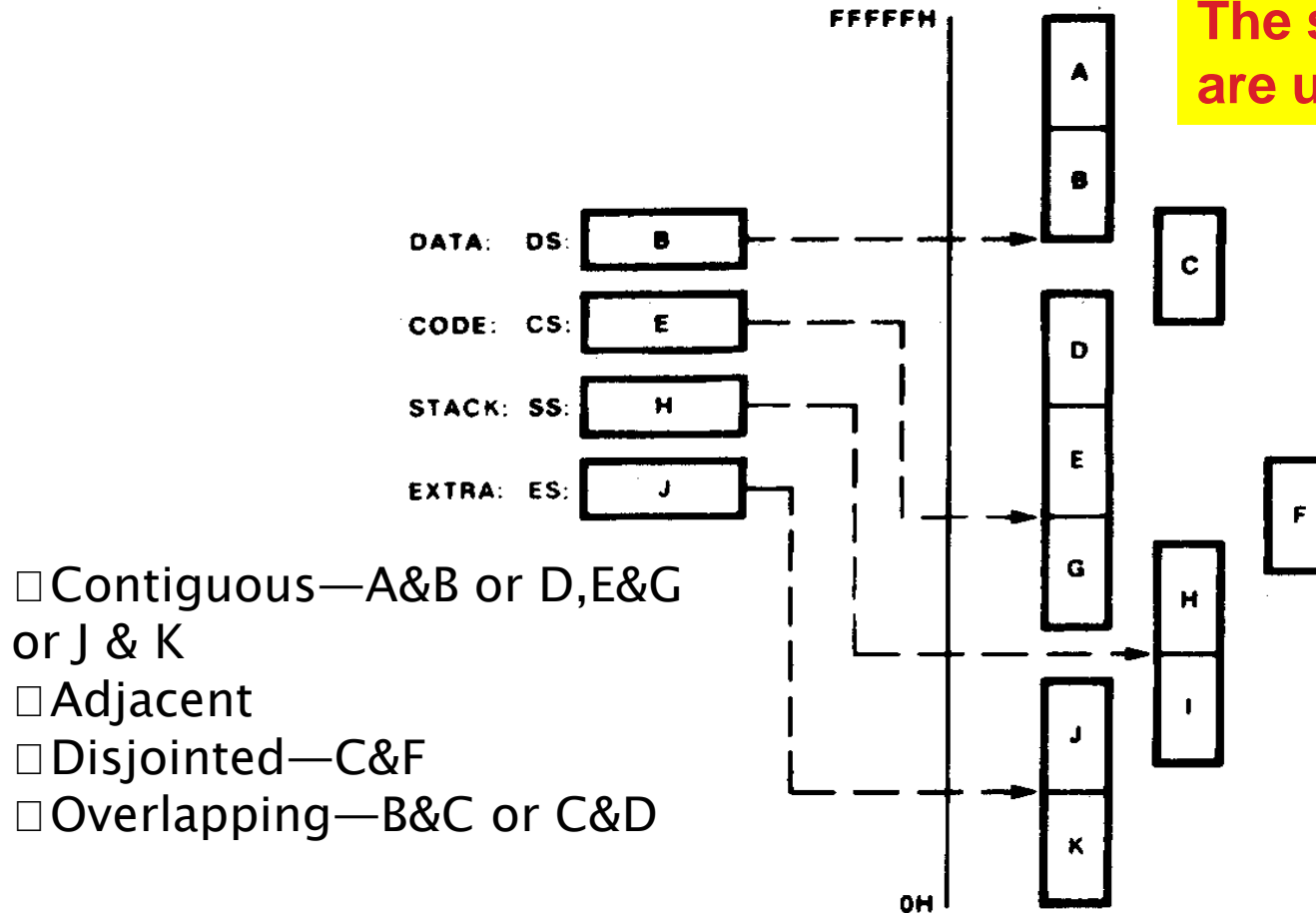
- ▶ A **segment** represents an independently **addressable** unit of memory consisting of **64K** consecutive byte wide storage locations.
- ▶ Each segment is assigned a **base address** that identifies its starting point.
- ▶ Only **four** segments can be **active** at a time:
 - The code segment
 - The stack segment
 - The data segment
 - The extra segment
- ▶ The addresses of the active segments are stored in the four internal **segment registers**: CS, SS, DS, ES.

2.5 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, disjointed, or even overlapping.

2.5 Segment Registers and Memory Segmentation

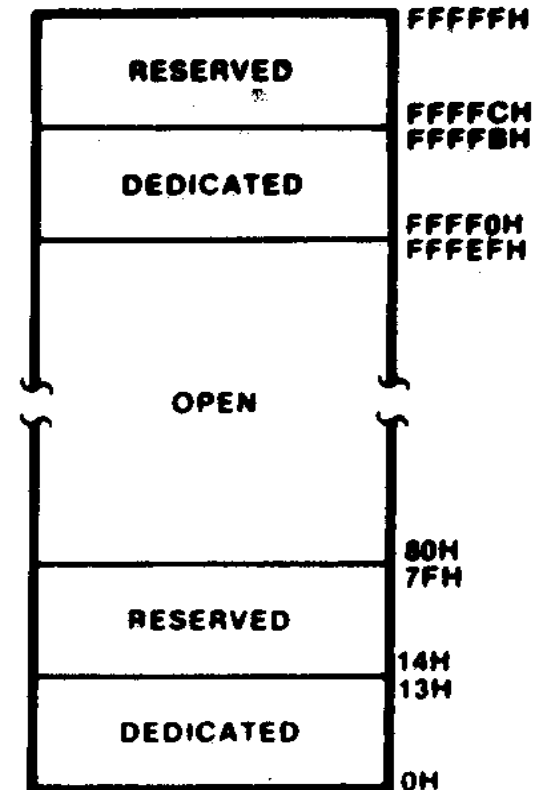
The segments registers are user accessible



Contiguous, adjacent, disjointed, and overlapping segments

2.6 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 (double word 4-bytes) ($128/4$).
- ▶ The **general-use memory** ($00080_{16} \sim FFFE_{16}$) stores **data** or **instructions** of the program.
- ▶ The **dedicated memory** ($FFFE0_{16} \sim FFFEB_{16}$) are used for hardware **reset jump** instruction.
- ▶ The **reserved memory** ($FFFCF_{16} \sim FFFFF_{16}$) are preserved for **future** use.



2.7 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.
- ▶ The offset in IP is combined with the current value in CS to generate the address of the instruction code. *CS:IP* forms 20-bit physical address of next word of instruction code.
- ▶ During normal operation, the 8088 fetches instructions from the code segment of memory, stores them in its instruction queue (*why?*), and executes them one after the other.

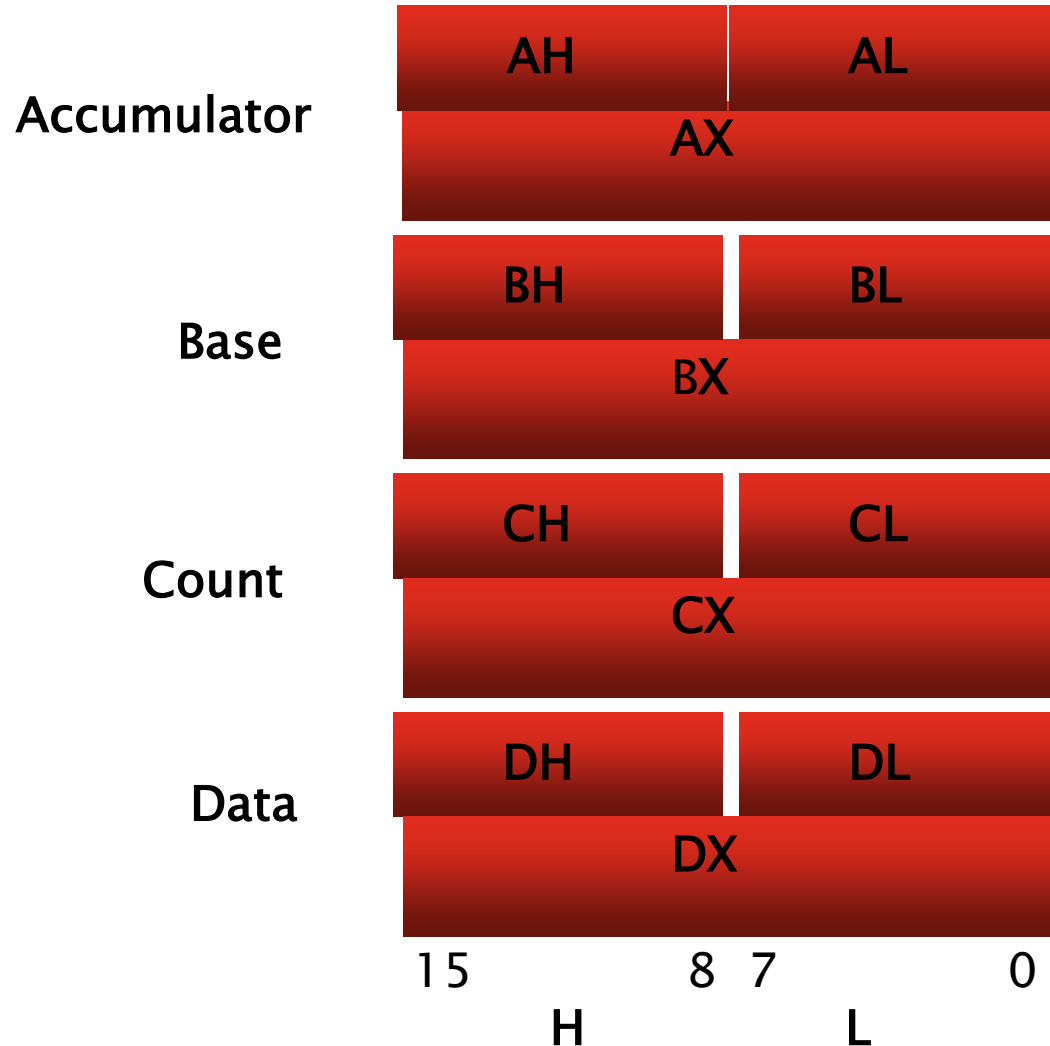
2.7 Instruction Pointer

- ▶ **Instruction fetch** sequence:
 - 8088/8086 fetches a word of instruction code from code segment in memory
 - Increments value in **IP** by **2**
 - Word placed in the instruction queue to wait for execution
 - 8088 prefetches up to **4** bytes of code
- ▶ **Instruction execution** sequence:
 - Instruction is read from output of instruction queue and executed
 - **Operands** read from data memory, internal registers
 - **Operation** specified by the instruction performed on operands
 - Result written back to either data memory or internal register

2.8 Data Registers

- ▶ The **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 data registers are:
 - **Accumulator** register, A
 - **Base** register, B
 - **Counter** register, C
 - **Data** register, D
- ▶ Each register can be accessed either as a whole (16 bits) for word data or as 8-bit data for byte-wide operation.

2.8 Data Registers



General-purpose data registers of 8088 microprocessor

2.8 Data Registers

- ▶ Uses:
 - Hold **data** such as source or destination operands for most operations—ADD, AND, SHL (**faster access**)
 - Hold **address** pointer for accessing memory
- ▶ Some also have dedicated special uses
 - C—count for loop, repeat string, shift, and rotate operations
 - B—Table look-up translations, base address
 - D—indirect I/O and string I/O

2.8 Data Registers

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

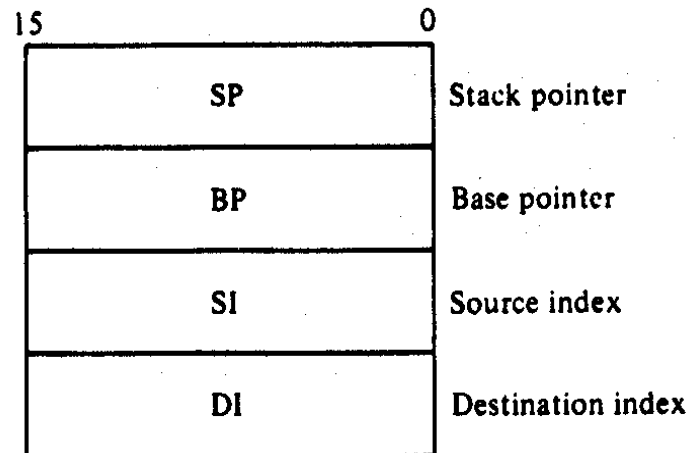
Dedicated register functions

2.9 Pointer and Index Registers

- ▶ 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.

2.9 Pointer and Index Registers

- ▶ The **index** registers 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.
 - DS:SI—points to source operand in data segment
 - DS:DI—points to destination operand in data segment
- ▶ Also used to access information in the extra segment (ES)
- ▶ The four registers must always be used for **16-bit** operations.

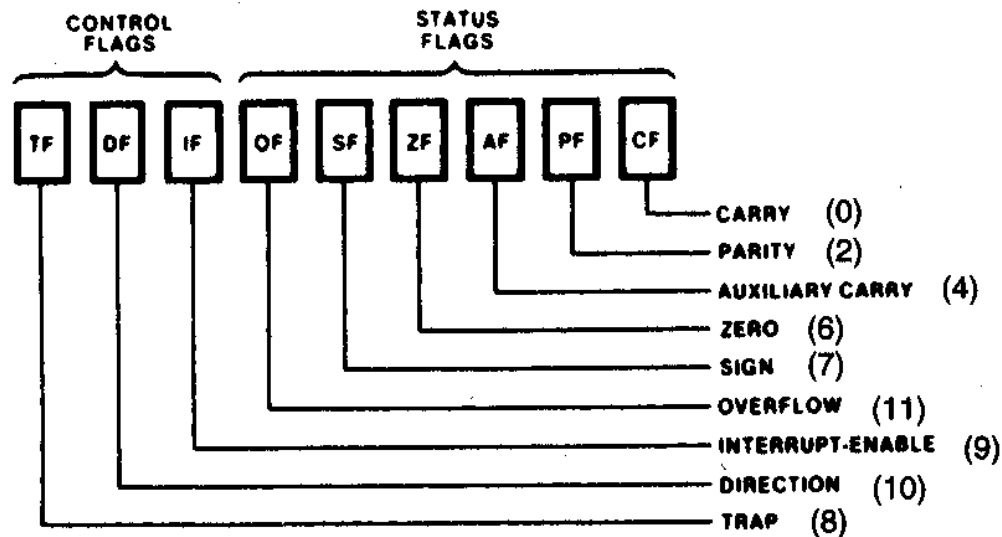


2.10 Status Register

- ▶ 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.

2.10 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

2.10 Status Register

- ▶ Status flags indicate current processor status.

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

Odd parity

2.10 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. Used to enable/disable external maskable interrupt requests
TF	Trap Flag	Allows Single-Step for debugging; causes interrupt after each op

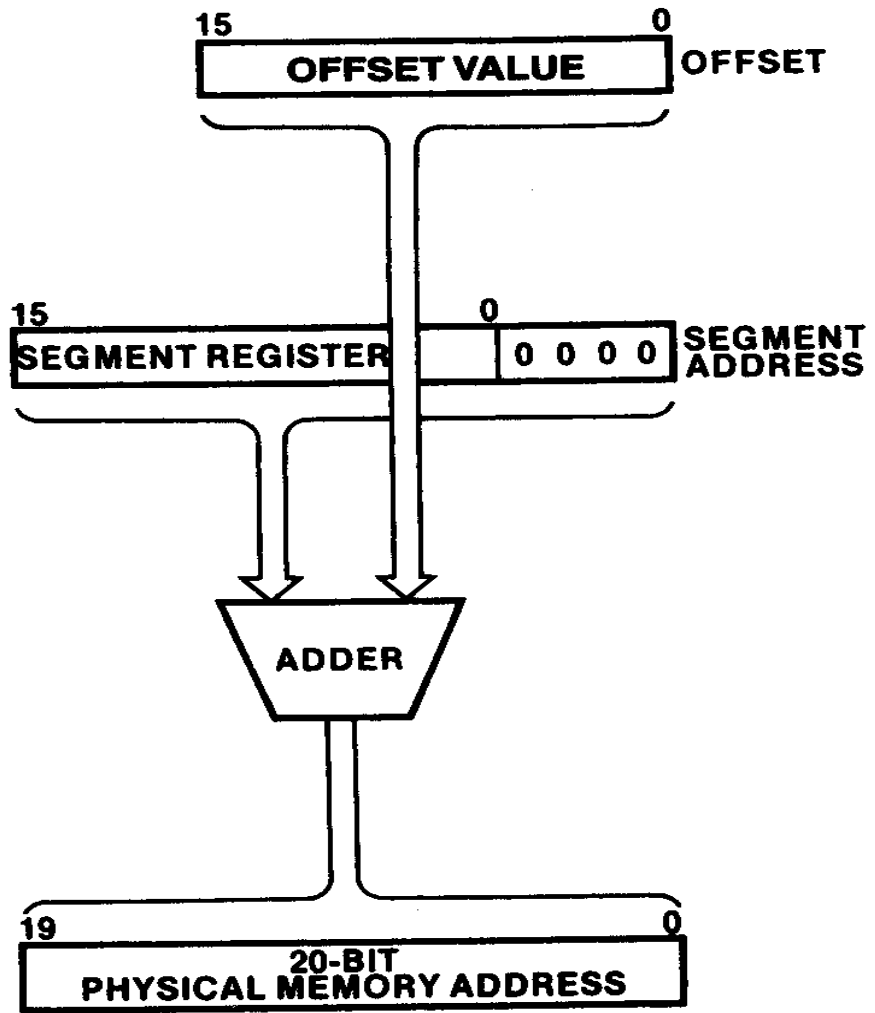
2.11 Generating a Memory Address

- ▶ A *logical address in the 8088 microcomputer* system is described by a segment base and an offset.
- ▶ 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.
- ▶ **Segment** base address (CS, DS, ES, SS) are 16 bit quantities.
- ▶ **Offsets** (IP, SI, DI, BX, DX, SP, BP, etc.) are 16 bit quantities.

2.11 Generating a Memory Address

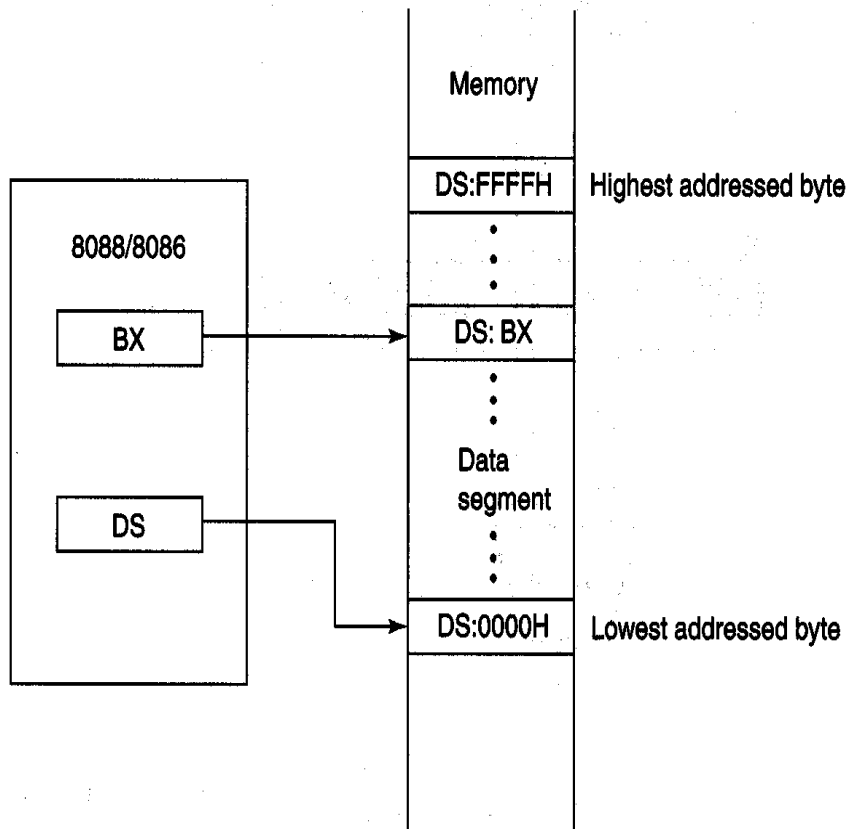
□ **Physical Address**: actual address used for accessing memory: 20-bits in length Formed by:

- Shifting the value of the 16-bit segment base address left 4 bit positions
- Filling the vacated four LSBs with 0s
- Adding the 16-bit offset



Generating a physical address

2.11 Generating a Memory Address



Boundary of a segment

- Four active segments CS, DS, ES, and SS
 - Each 64K-bytes in size → maximum of 256K-bytes of active memory
 - 64K-bytes for code
 - 64K-bytes for stack
 - 128K-bytes for data (data and extra)
 - **Starting** address of a data segment DS:0H → lowest addressed byte
 - **Ending** address of a data segment DS:FFFFH → highest addressed byte
 - Address of an element of data in a data segment
 - DS:BX → address of byte, word, or double word element of data in the data segment

2.11 Generating a Memory Address

□ Example:

Segment base address = 1234H

Offset = 0022H

1234H = 0001 0010 0011 0100₂

0022H = 0000 0000 0010 0010₂

Shifting base address,

00010010001101000000₂ = 12340H

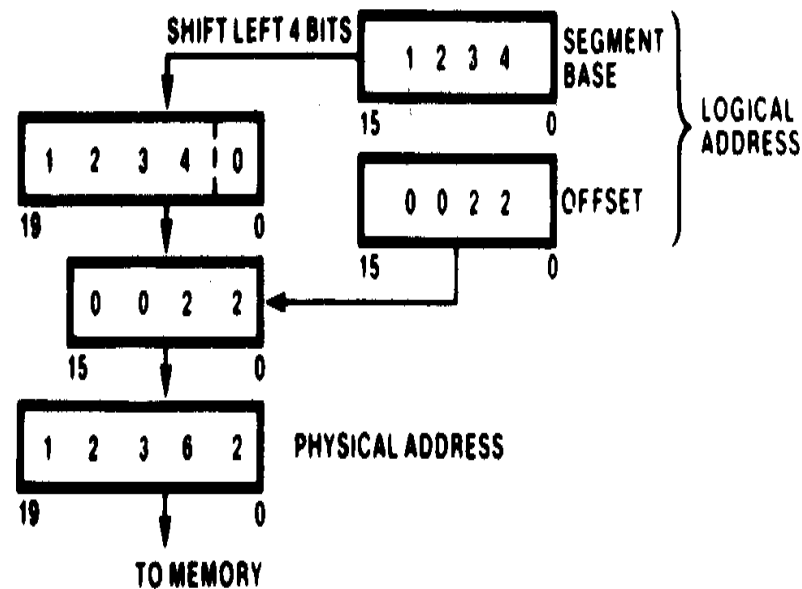
Adding segment address and offset

00010010001101000000₂ +

00000000000100010₂ =

= 00010010001101100010₂

= 12362H



2.11 Generating a Memory Address

▶ EXAMPLE

What would be the offset required to map to physical address location $002C3_{16}$ if the contents of the corresponding segment register are $002A_{16}$?

▶ Solution:

The offset value can be obtained by shifting the contents of the segment of the segment register left by four bit positions and then subtracting from the physical address. Shifting left give

$002A0_{16}$

Now subtracting, we get the value of the offset:

$$002C3_{16} - 002A0_{16} = 0023_{16}$$

2.1.1 Generating a Memory Address

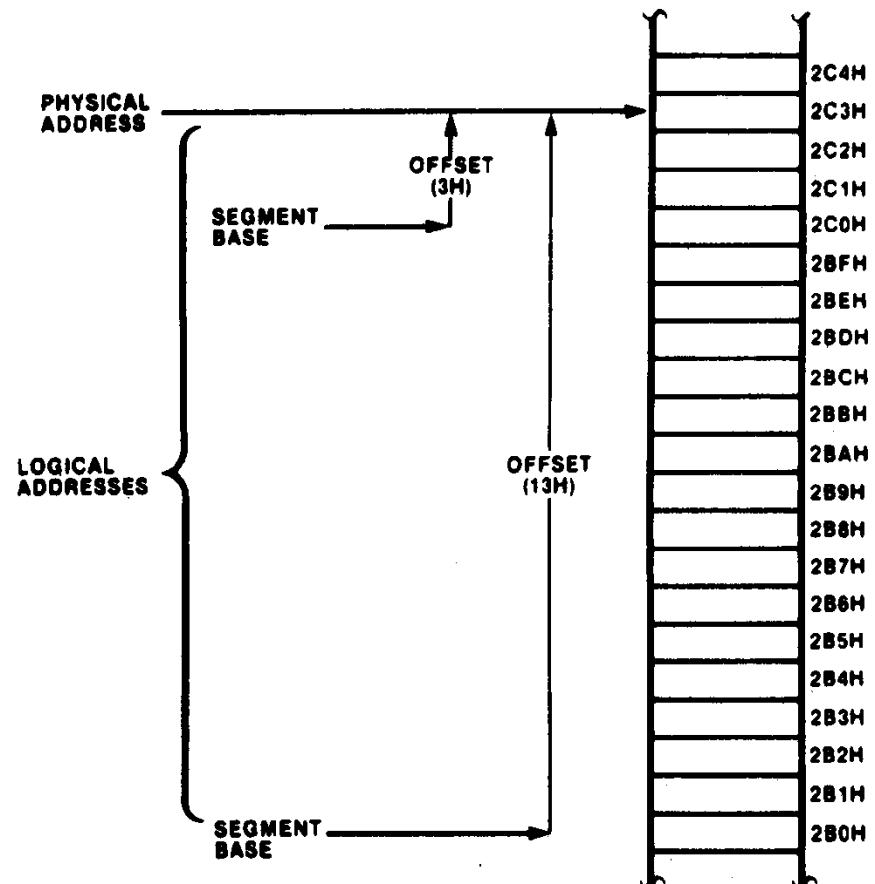
- ▶ **Different** logical addresses can be mapped to the **same** physical address location in memory.

□ Examples:

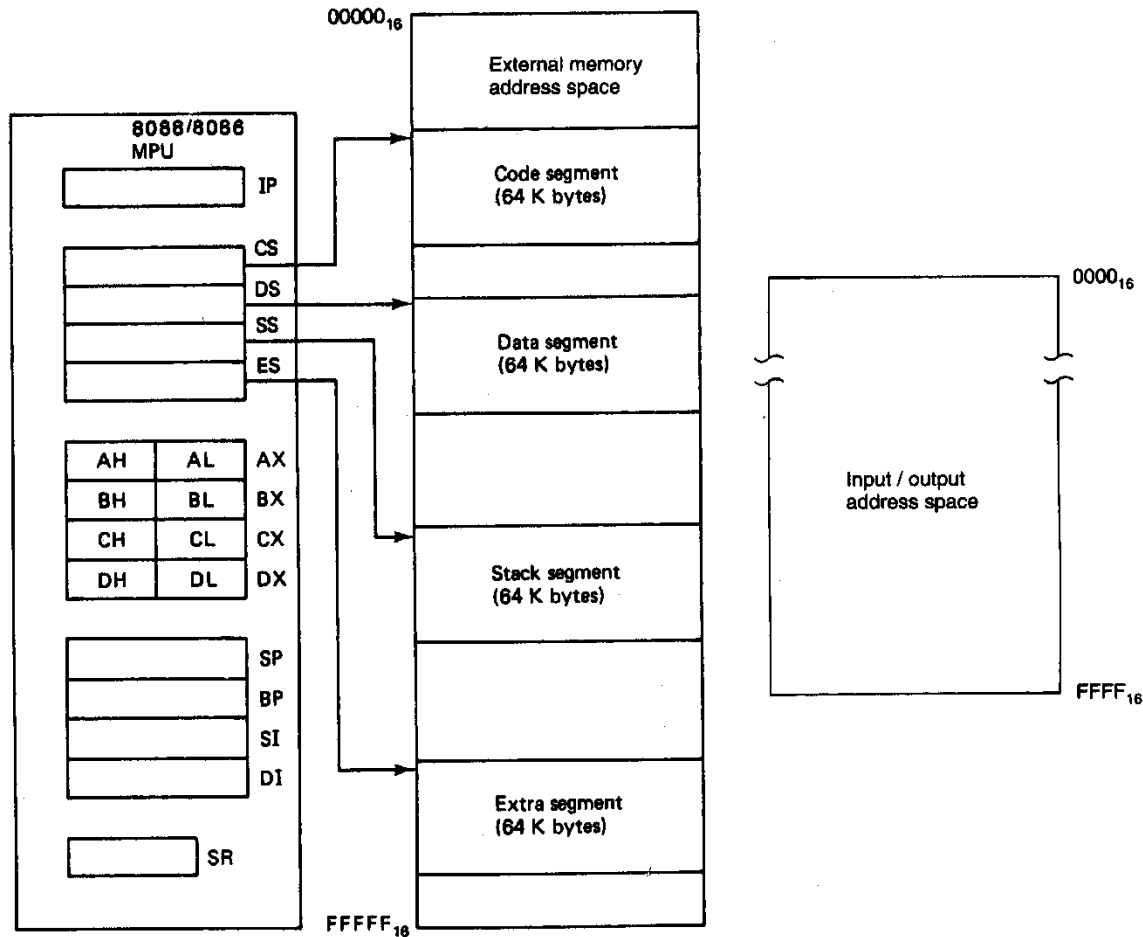
$2BH:13H = 002B0H + 0013H = 002C3H$

$2CH:3H = 002C0H + 0003H = 002C3H$

□ These logical addresses are called “**aliases**”



2.2 Software Model of the 8088/8086 Microprocessor



Software model of the 8088/8086 microprocessors

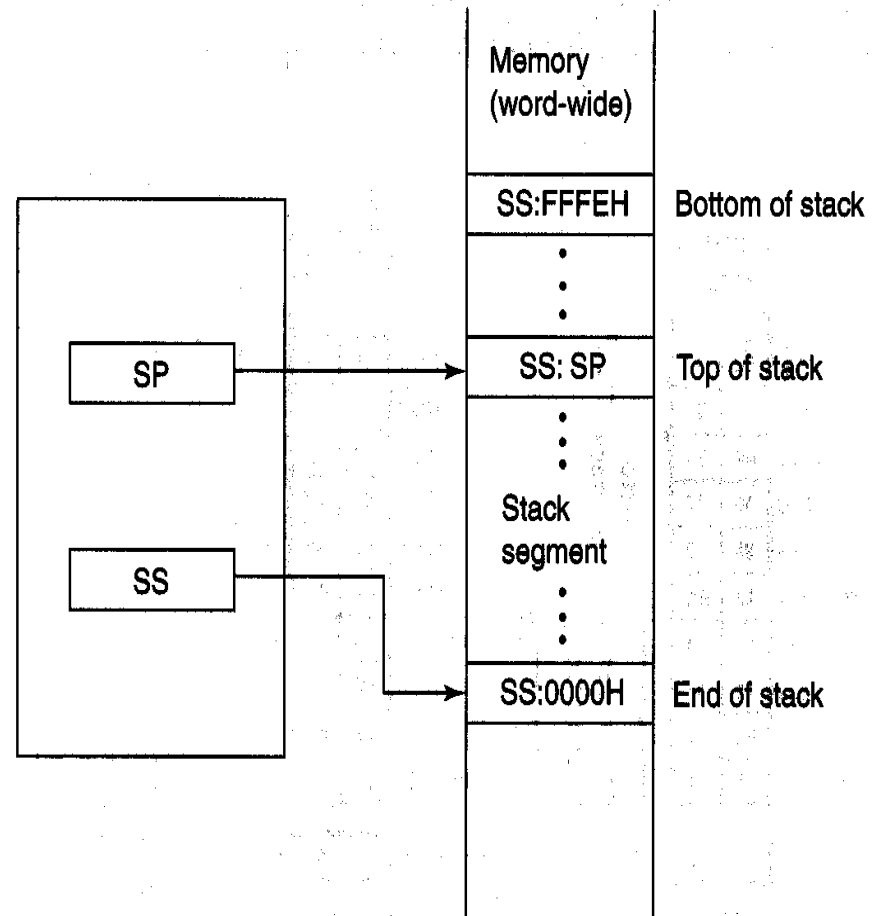
2.12 The Stack

- ▶ The **stack** is implemented for temporary storage of information such as data or addresses.
- ▶ The stack is **64KBytes** long and is organized from a software point of view as **32K words**.
- ▶ The contents of the **SP** and **BP** registers are used as offsets into the stack segment memory while the segment base value is in the **SS** register.
- ▶ Push instructions (**PUSH**) and pop instructions (**POP**)
- ▶ ***Top of the stack (TOS) and bottom of the stack (BOS)***
- ▶ The 8088 can push **word-wide data and address** information onto the stack from registers or memory.
- ▶ **Many stacks** can exist but only one is active at a time.

2.12 The Stack

Organization of stack:

- **SS:0000H** → end of stack (lowest addressed word)
- **SS:FFFEH** → bottom of stack (highest addressed word)
- **SS:SP** → top of stack (last stack location to which data was pushed)
- Stack grows down from higher to lower address
- Used by call, push, pop, and return operations
- Examples
 - PUSH SI** → causes the current content of the SI register to be pushed onto the “top of the stack”
 - POP SI** → causes the value at the “top of the stack” to be popped back into the SI register



Stack segment of memory

2.12 The Stack

□ Stack status prior to execution of the instruction **PUSH AX**:

AX = 1234H

SS = 0105H

AEOS = SS:00 → 01050H = end of stack

SP = 0008H

ABOS = 01050₁₆ + FFFE₁₆
= SS:FFEH → 1104EH

ATOS = 01050₁₆ + 0008₁₆
= SS:SP → 01058H = current top of stack

BBAAH = Last value pushed to stack

Addresses < 01058H = **invalid** stack data

Addresses ≥ 01058H = **valid** stack data

□ In response to the execution of PUSH AX instruction:

1. SP → 0006H decremented by 2

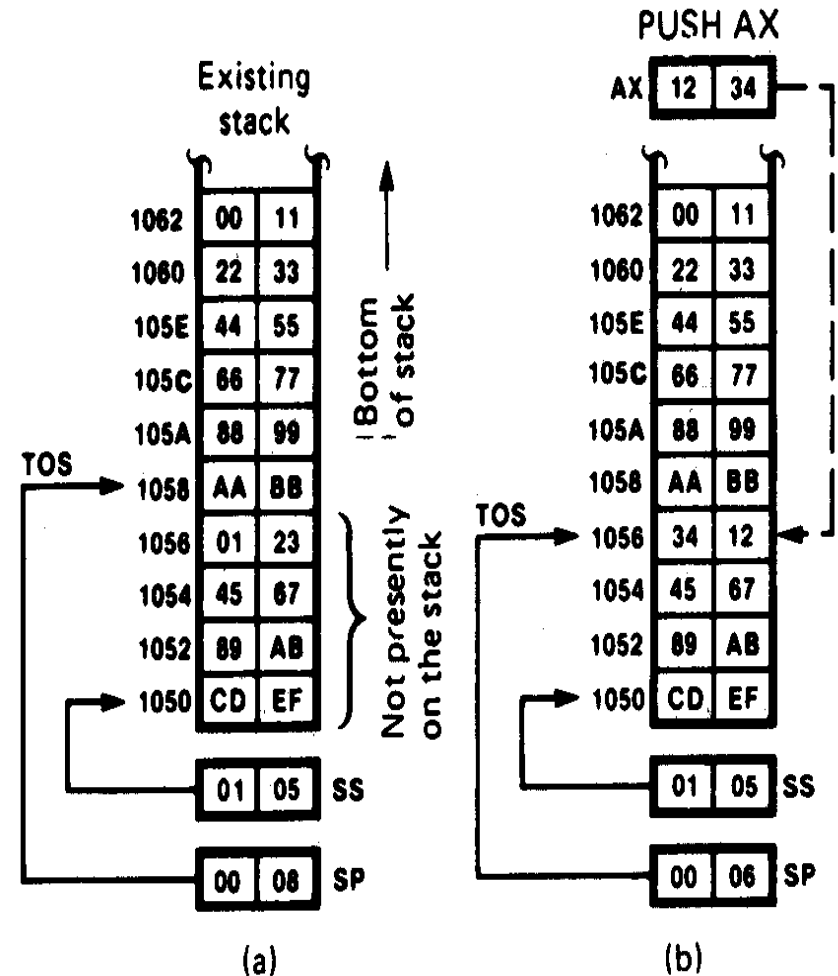
ATOP → 01056H

2. Memory write to stack segment

AL = 34H → 01056H

AH = 12H → 01057H

- How many free spaces ?
- When the stack get full ?



2.12 The Stack

• EXAMPLE: Pop operation

□ Status of the stack prior to execution of the instruction POP AX:

AX = XXXXH

SS = 0105H

SP = 0006H

ATOS = SS:SP → 01056H

1234H = Last value pushed to stack

Addresses < 01056H = invalid stack data

Addresses ≥ 01056H = valid stack data

□ In response to the execution of POP AX instruction

1. Memory read to AX

01056H = 34H → AL

01057H = 12H → AH

2. SP → 0008H incremented by 2

ATOP → 01058H

□ In response to the execution of POP BX instruction

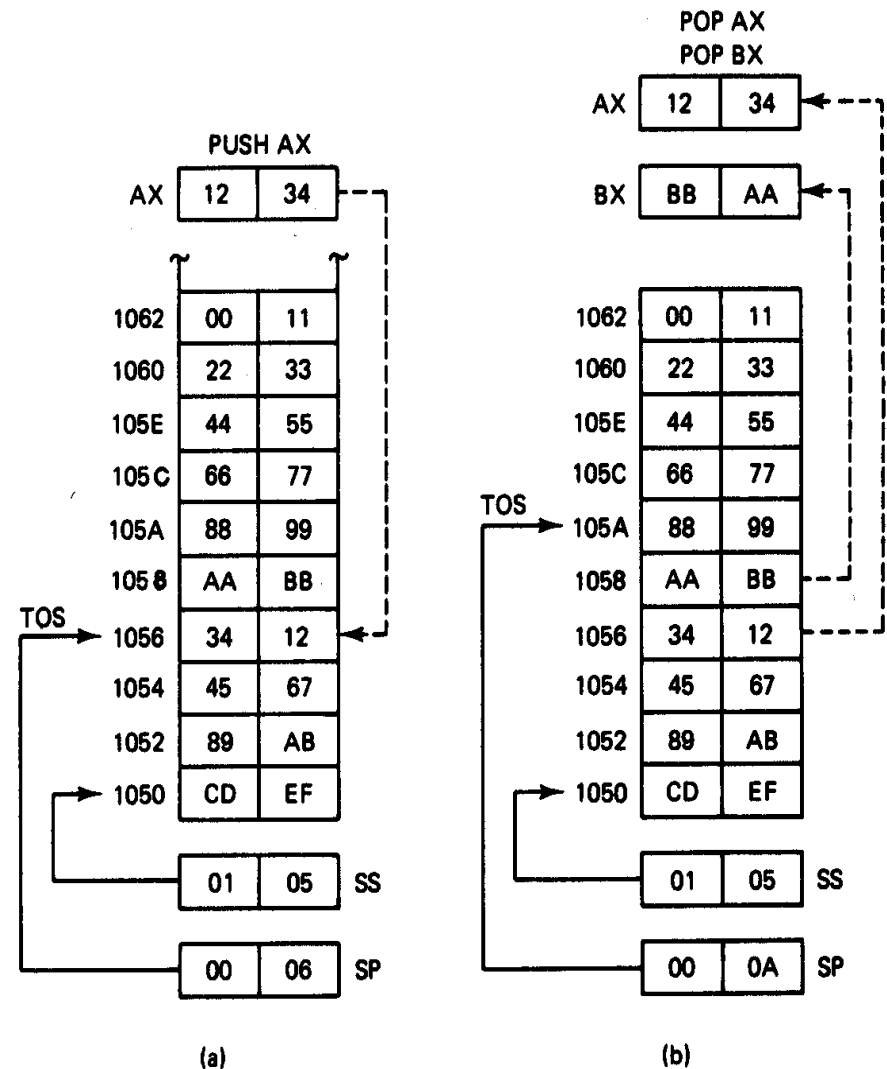
1. Memory read to BX

01058H = AAH → BL

01059H = BBH → BH

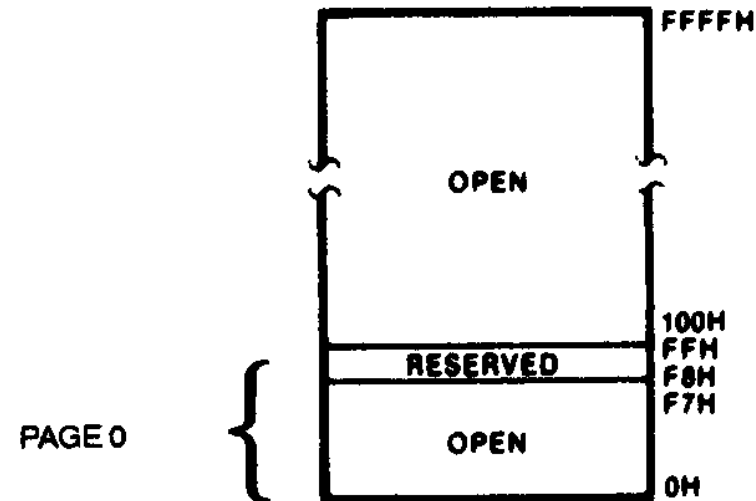
2. SP → 000AH incremented by 2:

ATOP → 0105AH

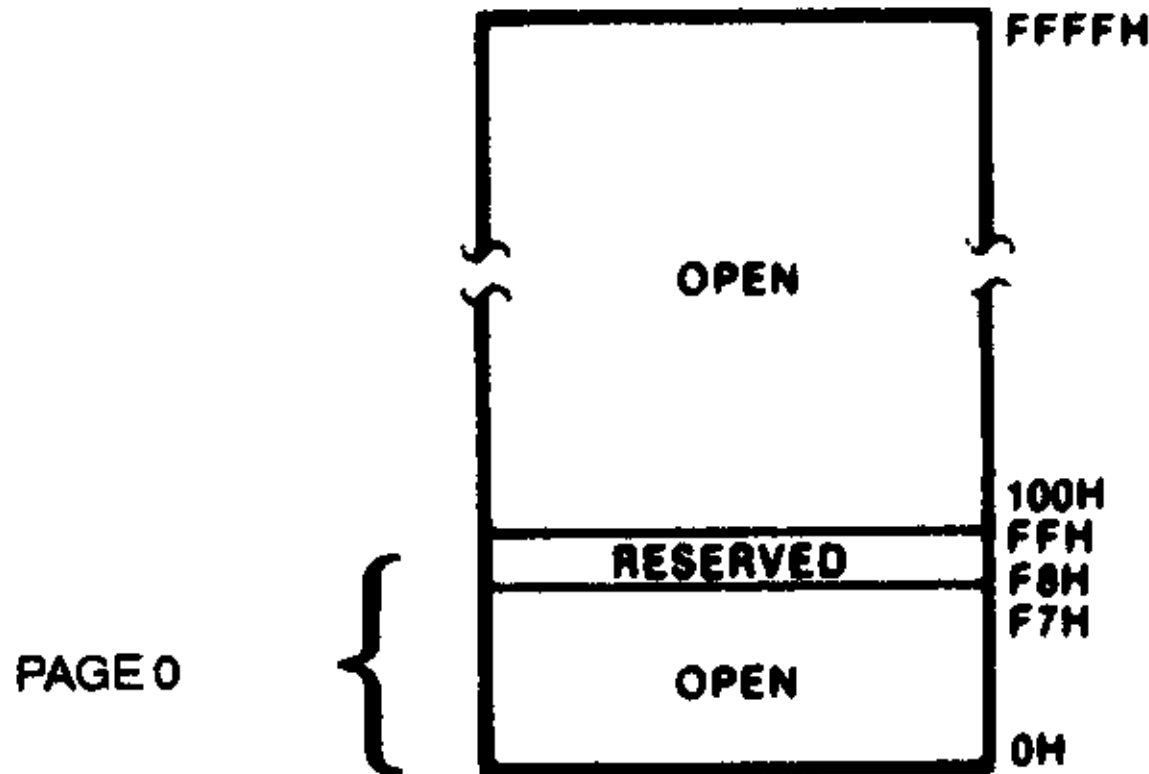


2.13 Input/Output Address Space

- ▶ The 8088 has separate memory and input/output (I/O) address space.
- ▶ The I/O address space is the place where I/O interfaces, such as printer and terminal ports, are implemented.
- ▶ The I/O address range is from 0000_{16} to $FFFF_{16}$. This represents **64KByte** addresses.
- ▶ The I/O addresses are **16** bits long. Each of these addresses corresponds to one byte-wide I/O port.
- ▶ Certain I/O instructions can only perform operations to addresses 0000_{16} thru $00FF_{16}$ (**page 0**).
- ▶ Ports F8H through FF **reserved**



2.13 Input/Output Address Space



I/O address space

H.W. #2

- ❑ Solve the following problems from Chapter 2 from the course textbook:

3, 9, 14, 19, 26, 32, 34, 37, 45, 49, 55, 60, 65

If $SS = C000$, $SP = FF00$

- 1) How many data words currently in the stack
- 2) How the value $EE11$ will be pushed in the stack