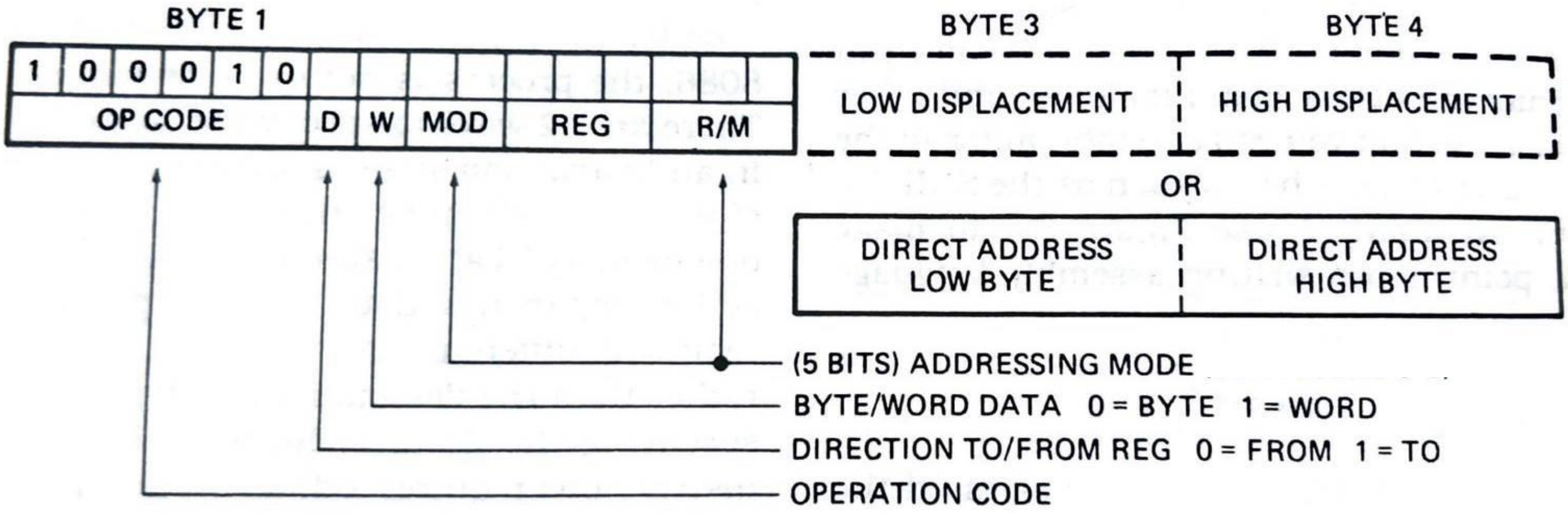
BITS Pilani

Microprocessors & Interfacing

INSTRUCTION SET

Dr. Gargi Prabhu
Department of CS & IS

Instruction Format



Direction Bit

D=1 , if data is moved **TO register** identified by REG field

D=0, if instruction is moving data **FROM register** mentioned in REG field

e.g. 8BECH

Opcode						D	W
1	0	0	0	1	0	1	1

MOD		REG			R/M		
1	1	1	0	1	1	0	0

Opcode = MOV
D = Transfer to register (REG)
W = Word
MOD = R/M is a register
REG = BP
R/M = SP

Since D=1, data is moved to BP (101) from SP (100)

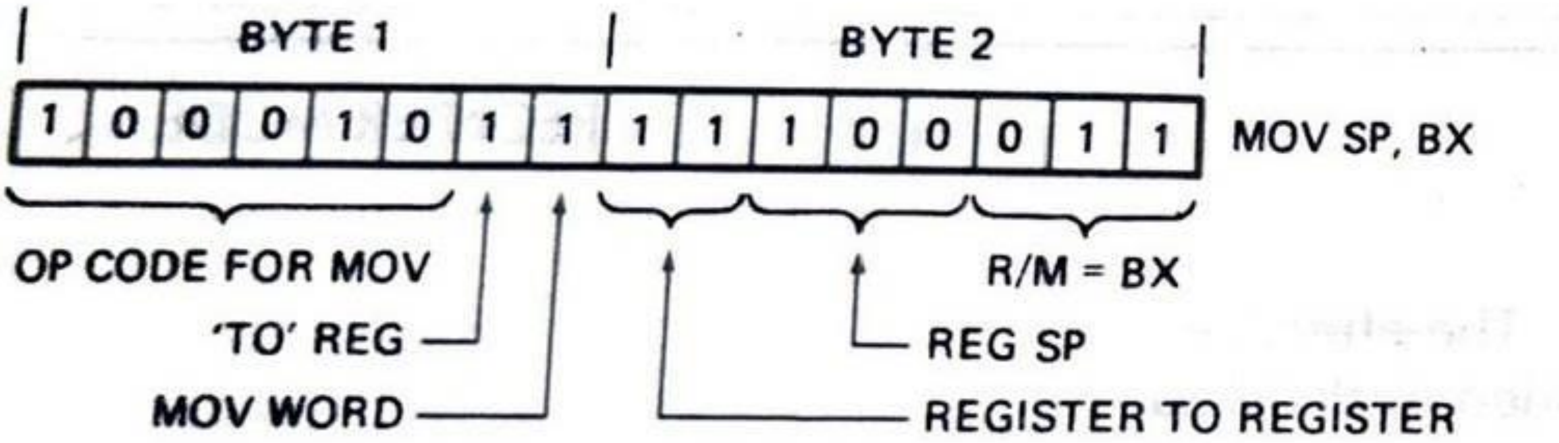
MOV BP,SP

Direction Bit

D=1 , if data is moved **TO register** identified by REG field

D=0, if instruction is moving data **FROM register** mentioned in REG field

e.g. MOV SP,BX



Direction Bit

D=1 , if data is moved **TO register** identified by REG field

D=0, if instruction is moving data **FROM register** mentioned in REG field

e.g. MOV SP,BX

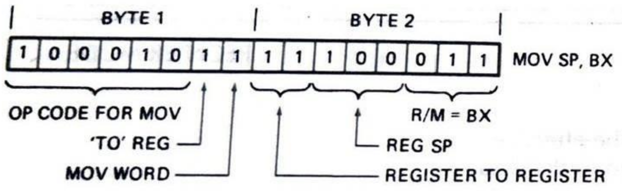
What if D=0 ?

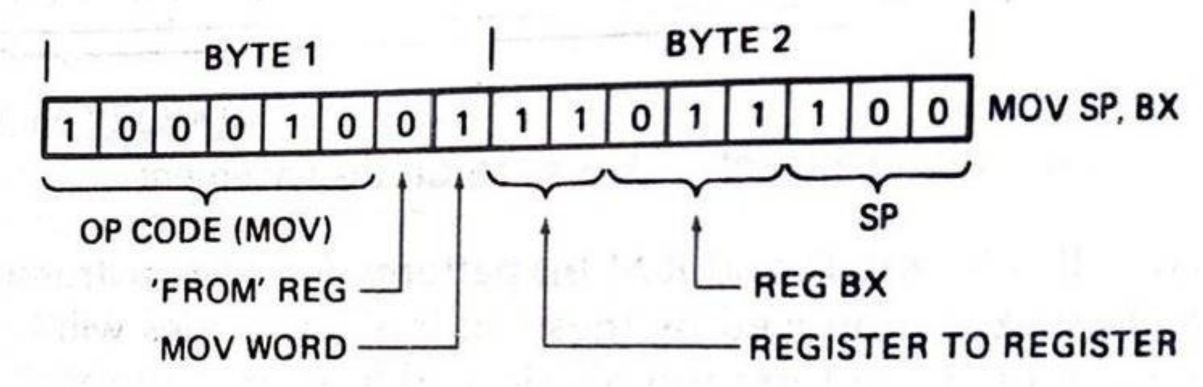
Direction Bit

D=1 , if data is moved **TO register** identified by REG field

D=0, if instruction is moving data **FROM register** mentioned in REG field

e.g. MOV SP,BX



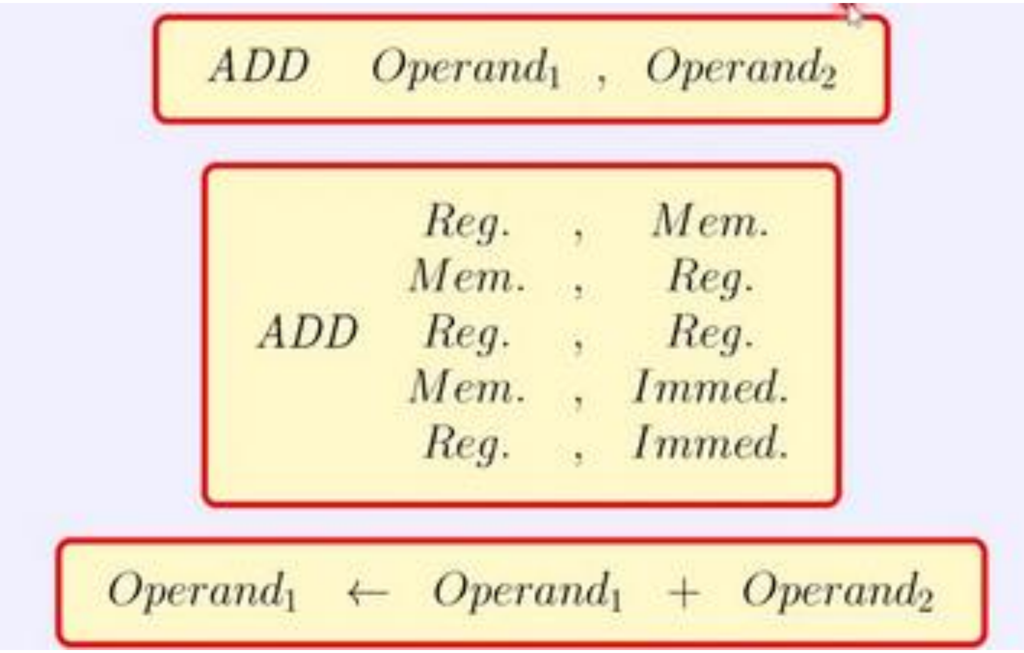


- Which MOV is used in 8086? D=0 in Intel Macroassembler (Ref: Douglas V. Hall)

Example: Base-relative-plus-index

What is the machine code for ADD [BX+DI+1234H], AX ?

Opcode=000000



$$ADD \quad Operand_1 \; , \; Operand_2$$

$$ADD \quad \begin{array}{ccc} Reg. & , & Mem. \\ Mem. & , & Reg. \\ Reg. & , & Reg. \\ Mem. & , & Immed. \\ Reg. & , & Immed. \end{array}$$

$$Operand_1 \leftarrow Operand_1 + Operand_2$$

Output of ADD instruction will be stored in memory address hence D=0. AX is source operand (REG). Since data is moving from REG, D=0.

Example: Base-relative-plus-index

What is the machine code for ADD [BX+DI+1234H], AX ?

Opcode=000000

D=0 since AX is source operand

W=1 ->16 bit data operation

Byte1=(00000001) = 01H

MOD= 10 (Memory mode with 16-bit displacement)

REG=000 (Code of AX)

R/M=001 (BX+DI)

Byte2=(10000001)=81H

Displacement=1234H -> stored as LSB MSB

Machine code = 01813412

Output of ADD instruction will be stored in memory address hence D=0. AX is source operand (REG). Since data is moving from REG, D=0.

Example

What is the machine code for MOV CS:[BX], DL ?

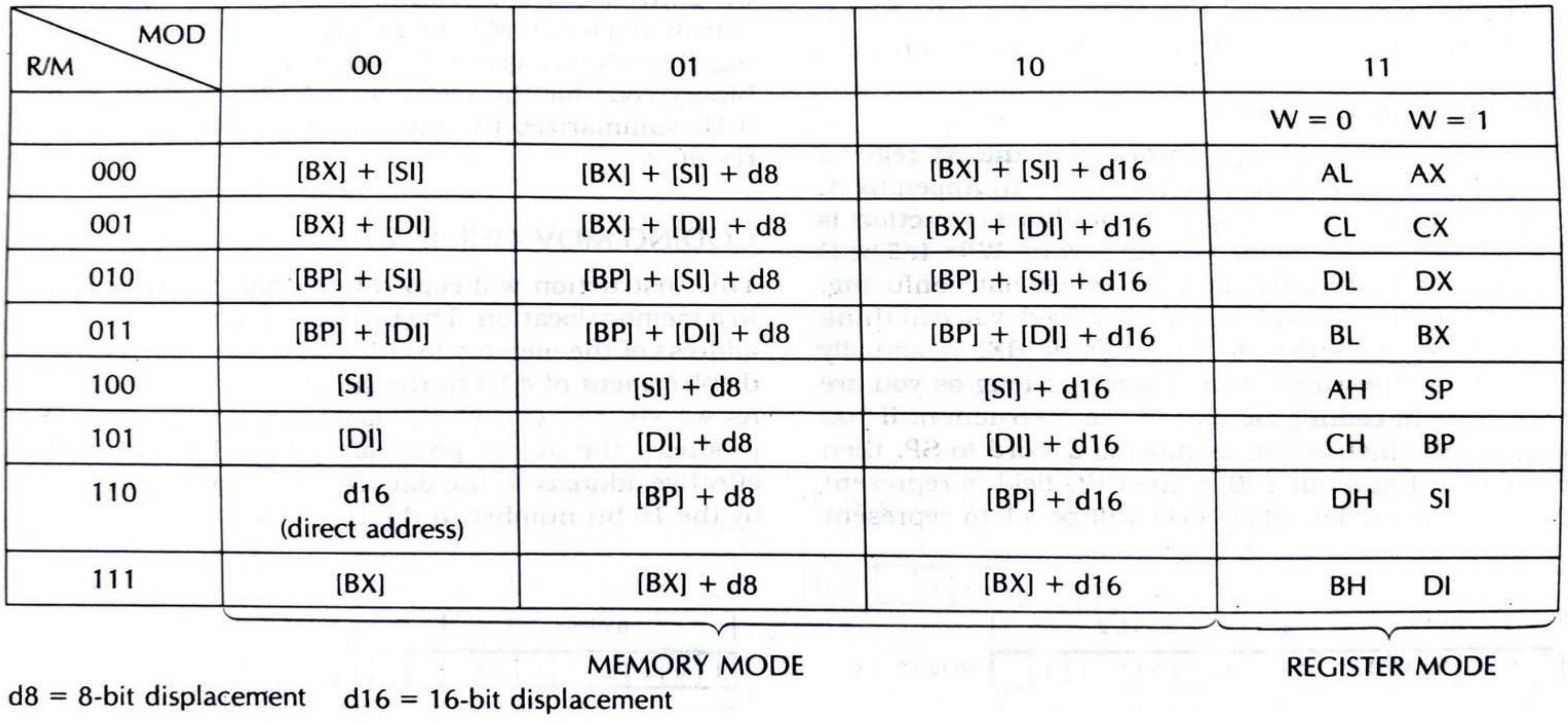
- In this instruction, CS: indicates Physical Address =BX+CS*10H

- CS: Segment Override Prefix

- Code byte for Segment Override Prefix = 001XX110 which forms the first byte of instruction.

- XX-> ES=00, CS=01, SS=10, DS=11

Instruction Format



R/M \ MOD	00	01	10	11	
				W = 0	W = 1
000	[BX] + [SI]	[BX] + [SI] + d8	[BX] + [SI] + d16	AL	AX
001	[BX] + [DI]	[BX] + [DI] + d8	[BX] + [DI] + d16	CL	CX
010	[BP] + [SI]	[BP] + [SI] + d8	[BP] + [SI] + d16	DL	DX
011	[BP] + [DI]	[BP] + [DI] + d8	[BP] + [DI] + d16	BL	BX
100	[SI]	[SI] + d8	[SI] + d16	AH	SP
101	[DI]	[DI] + d8	[DI] + d16	CH	BP
110	d16 (direct address)	[BP] + d8	[BP] + d16	DH	SI
111	[BX]	[BX] + d8	[BX] + d16	BH	DI

MEMORY MODE

REGISTER MODE

d8 = 8-bit displacement   d16 = 16-bit displacement

Example: MOV CS:[BX], DL

Byte 1 = 00101110

Opcode =100010

D= from REG = 0

W= 0

Byte 2 = 10001000

MOD=00 (Memory, No Displacement)

REG=010 (DL)

R/M = 111 ([BX])

Byte 3= 00010111

Example: MOV [1000H],DL

MOD=00

REG=010

R/M=110

D=0 -> Transfer from REG DL

Byte1 = 100010 0 0

Byte2 = 00 010 110

Byte3 = 0000 0000 (Displacement – LOW)

Byte4 = 0001 0000 (Displacement – HIGH)

Data Movement Instructions

MOV Instructions available:

- MOV
- MOVSX -**Move with Sign-Extend**
- MOVZX -**Move with Zero-Extend**

- When do you use MOVSX and MOVZX?

- Need to work with values of different sizes (e.g., moving from a byte to a word) while preserving the sign (for MOVSX) or not (for MOVZX).

Data Movement Instructions

MOVSX -**Move with Sign-Extend**

- This instruction copies the source operand into the destination operand, while sign-extending the source to fill the destination.

e.g. MOVSX AX, AL

MOVZX -**Move with Zero-Extend**

- Copies the source operand into the destination operand. However, in this case, zero-extension is performed instead of sign-extension.

e.g. MOVZX AX, AL

Stack Memory Instructions

Two Instructions

- PUSH
- POP

Six forms of PUSH and POP instructions: register, memory, immediate, segment register, flags and all registers.

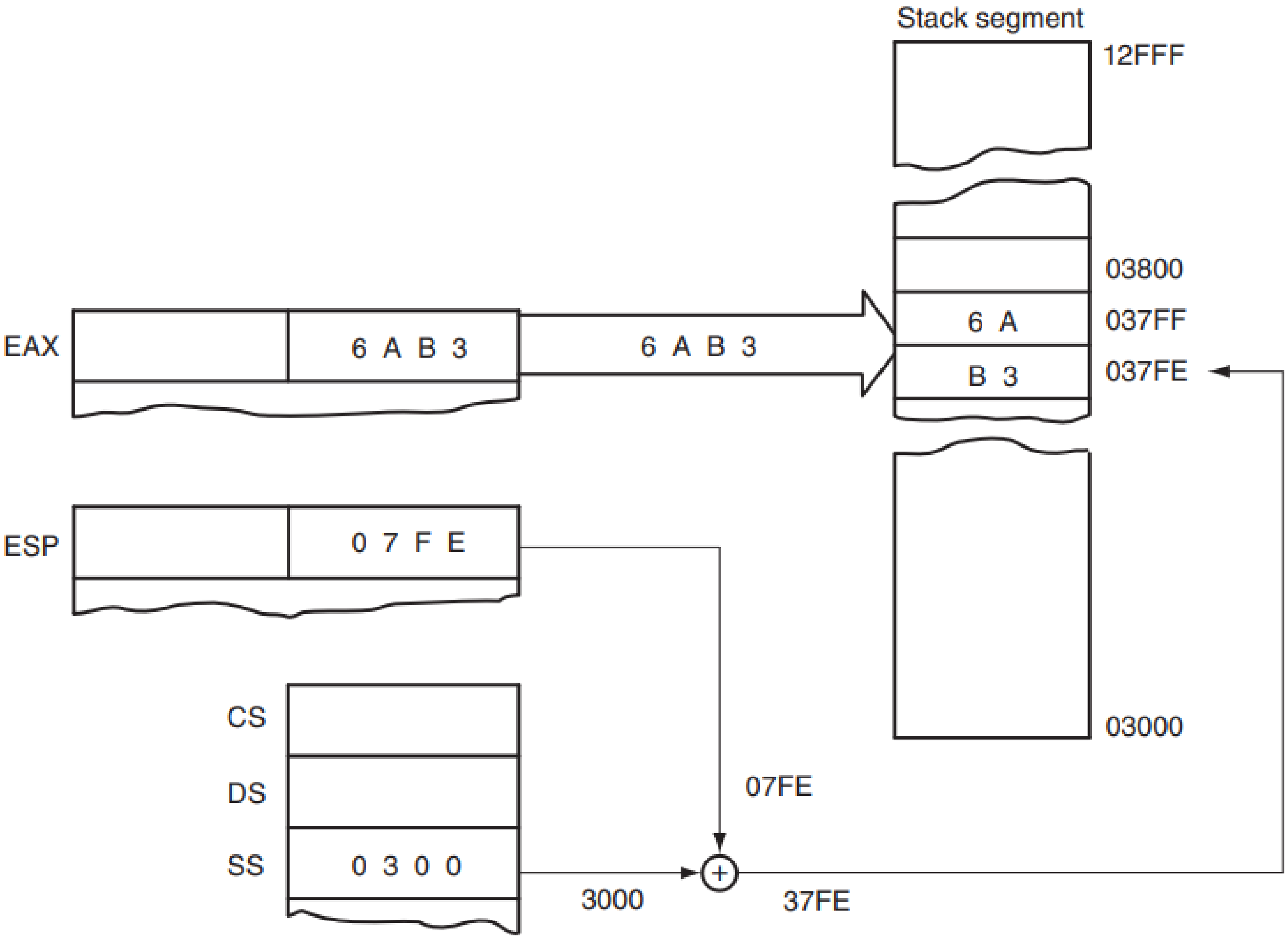
Note that PUSH and POP immediate and PUSHA and POPA (all registers) are not available in 8086, but are available 80286 onwards.

PUSH Instruction

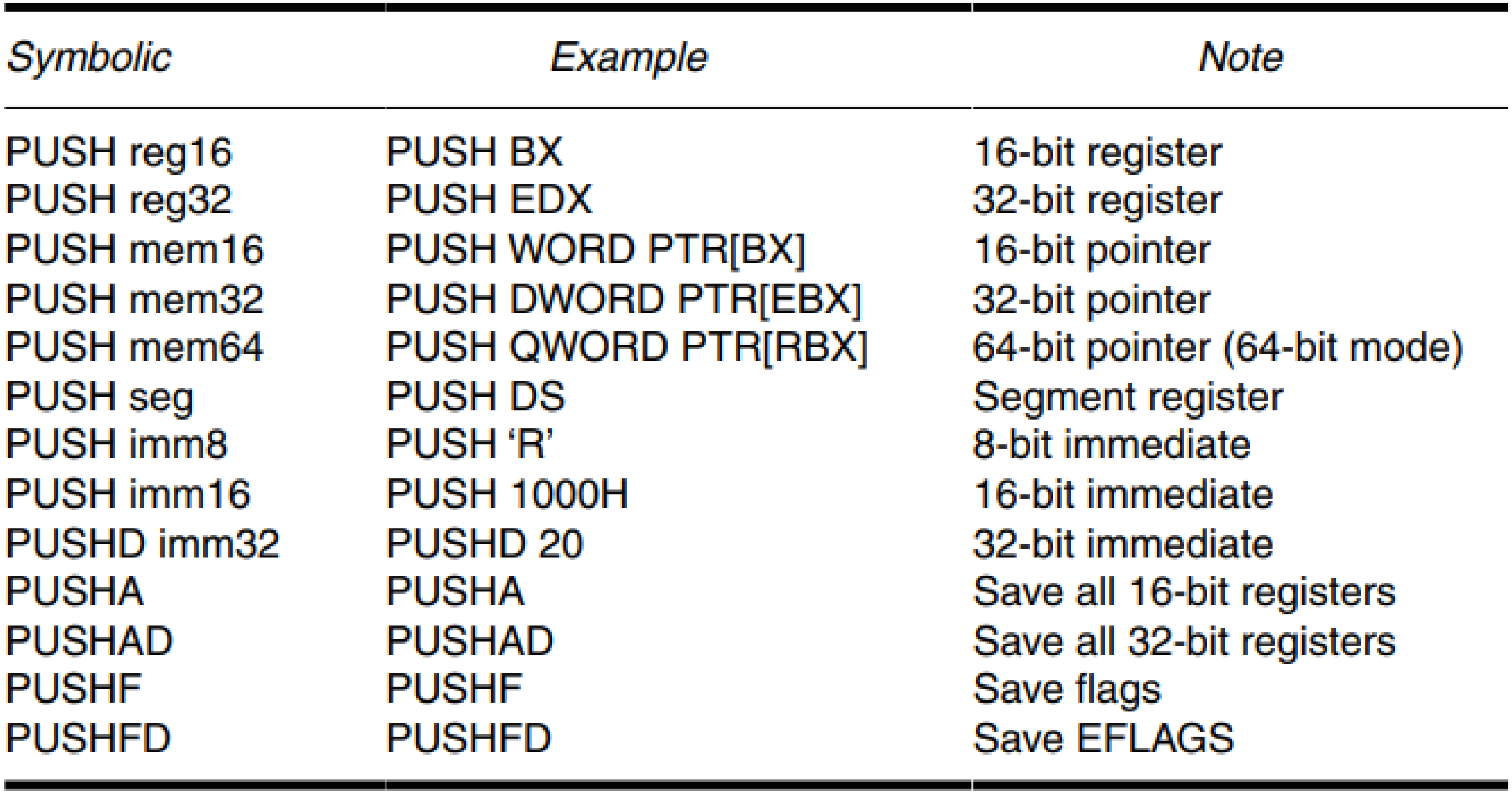
- Transfers 2 bytes of data to the stack-> **PUSH** and **POP** work with 16 bit values only!
- Source: any internal 16 bit register, immediate data, any segment register, any 2 bytes of memory data
- When data are pushed onto the stack, the first(most-significant) data byte moves to the stack segment memory location addressed by SP-1
- Second data byte moves to SP-2
- After the two PUSH operations, contents of SP= decrement by 2

The 8086 architecture was designed with a focus on 16-bit data processing, and as a result, the stack-related instructions, including push and pop, work with 16-bit registers. Attempting to use these instructions directly with 8-bit registers would result in unexpected behavior or errors.

PUSH Instruction



PUSH Instructions

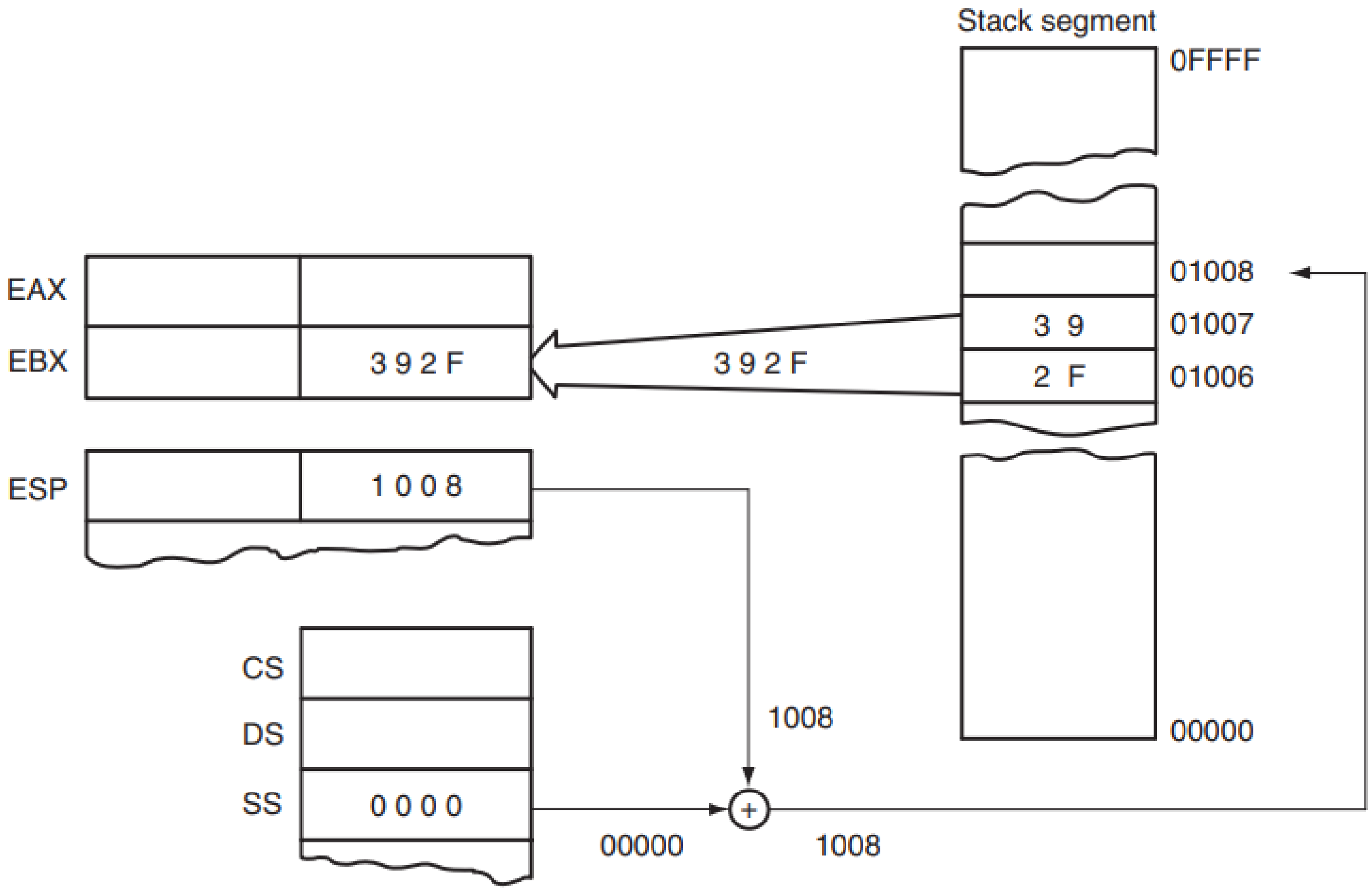


Symbolic	*Example*	*Note*
PUSH reg16	PUSH BX	16-bit register
PUSH reg32	PUSH EDX	32-bit register
PUSH mem16	PUSH WORD PTR[BX]	16-bit pointer
PUSH mem32	PUSH DWORD PTR[EBX]	32-bit pointer
PUSH mem64	PUSH QWORD PTR[RBX]	64-bit pointer (64-bit mode)
PUSH seg	PUSH DS	Segment register
PUSH imm8	PUSH 'R'	8-bit immediate
PUSH imm16	PUSH 1000H	16-bit immediate
PUSHD imm32	PUSHD 20	32-bit immediate
PUSHA	PUSHA	Save all 16-bit registers
PUSHAD	PUSHAD	Save all 32-bit registers
PUSHF	PUSHF	Save flags
PUSHFD	PUSHFD	Save EFLAGS

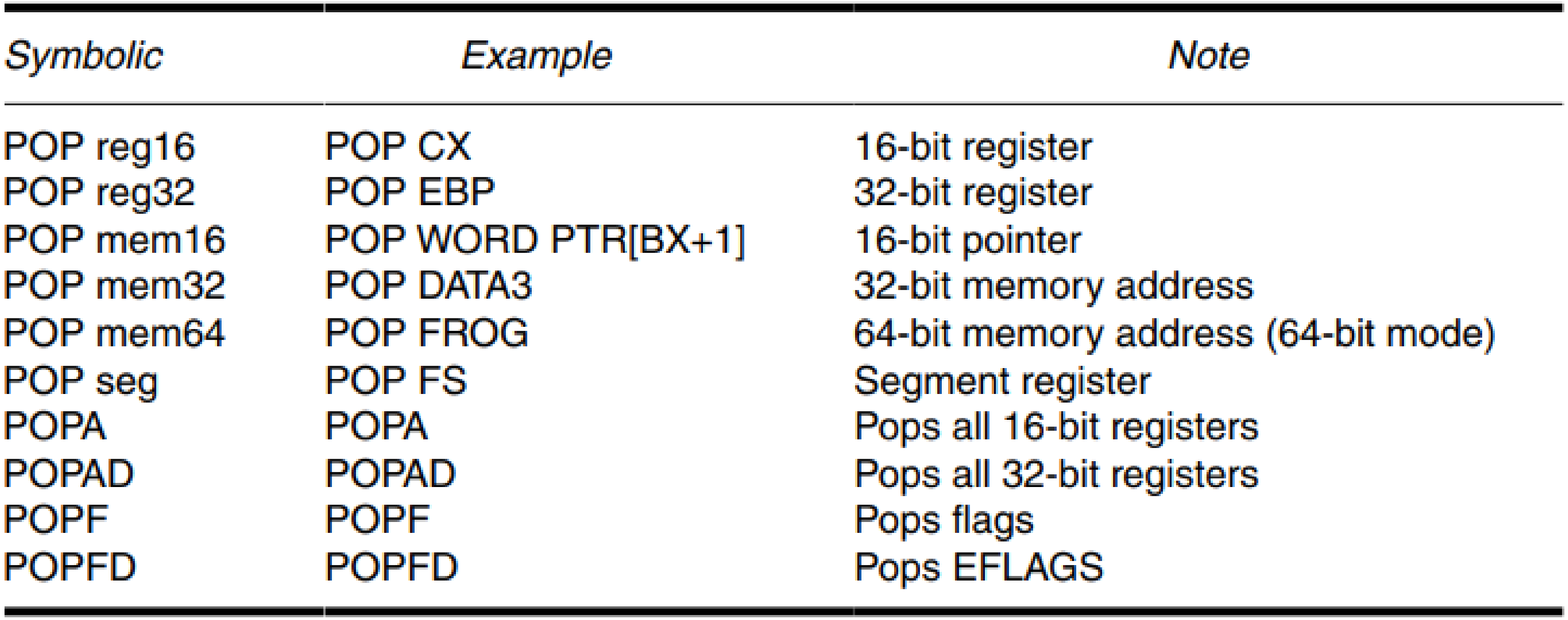
POP Instruction

- Performs the inverse operation of a PUSH instruction
- Removes data from the stack and places it into 16-bit target register, segment register, or a 16 bit memory location
- POPF-removes 16-bit numbers from stack and places it into the flags register

POP Instruction



POP Instructions



Symbolic	*Example*	*Note*
POP reg16	POP CX	16-bit register
POP reg32	POP EBP	32-bit register
POP mem16	POP WORD PTR[BX+1]	16-bit pointer
POP mem32	POP DATA3	32-bit memory address
POP mem64	POP FROG	64-bit memory address (64-bit mode)
POP seg	POP FS	Segment register
POPA	POPA	Pops all 16-bit registers
POPAD	POPAD	Pops all 32-bit registers
POPF	POPF	Pops flags
POPFD	POPFD	Pops EFLAGS

innovate achieve lead

BITS Pilani
Pilani Campus

Thank You