Topic-IV

80X86 Instruction Sets and ALP

T1. Barry B Brey, The Intel Microprocessors .Pearson, Eight Ed. 2009. Chapter 4-6, 8

Feb 9th 2021

Types of Instructions

- Instructions with two operands (source and destination)
  -(R R, R M, R Idata, M Idata, but not M M)
- Instructions with one operand (source or destination)
  -(R , M, but not Idata)
- Instructions without any operand

Types of Instructions

- Data Transfer Instructions
- Arithmetic Instructions
- Logical Instructions
- Branch and Program control Instructions

Data Transfer Instructions

1. Data Transfer Instructions

- MOV destination, source
- XCHG destination, source
- XLAT
- PUSH source
- POP destination
- IN Reg, Port address — Input the Port
- OUT Port address , Reg — Output to the port
- LEA 16 bit register, memory
- LDS 16 bit register, memory
- LES 16 bit register, memory
- LAHF
- SAHF
- PUSHF
- POPF

1. Data Transfer Instructions

- General Purpose Data Transfer

  (MOV, XCHG, XLAT, PUSH, POP)
- Input / Output Data Transfer

  (IN, OUT)
- Address Object Data Transfer

  (LEA, LDS, LES)
- Flag Transfer Data Transfer

  (LAHF, SAHF, PUSHF, POPF)

1. Data Transfer Instructions

i. General Purpose Data Transfer

- MOV : Move byte or word
- XCHG : Exchange byte or word
- XLAT : Translate byte
- PUSH: Push word onto stack
- POP : Pop word from stack

1. Data Transfer Instructions

- **MOV DST, SRC**

  - Copies the content of source to destination
  - No Flags Affected
  - Size of source and destination must be the same
  - Source can be register, memory, or immediate data
  - Destination can be register or memory location

Different MOV options

R ← M

M ← R

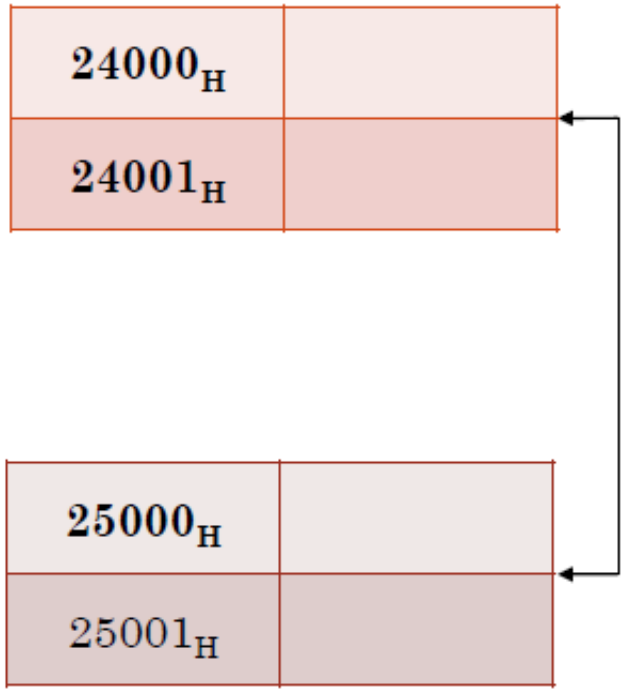
R ← R

M ← I

R ← I

MOV Instructions

Example 1: Swap the word at memory location 24000_H with 25000_H

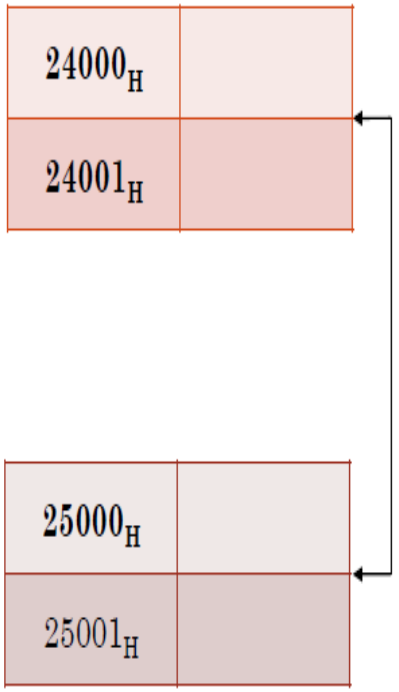


MOV Instructions

```
MOV AX, 2000H
MOV DS, AX
MOV SI, 4000H
MOV DI, 5000H
MOV BX, [SI]
MOV DX, [DI]

MOV [SI], DX
MOV [DI], BX
```

- Initialise Segment Register
- Initialise Offset Registers
- Transfer data from reg to mem temporarily
- Store back the data in mem



MOV Instructions

Example 2:

MOV BX, 2000_H
MOV DI, 10_H
MOV AL, [BX+DI]
MOV DI, 20_H
MOV [BX+DI], AL

DS: 2020 ← DS: 2010

XCHG Instructions

- XCHG : (exchange)switches the contents of the source and destination operands (byte or word).
- Can not exchange the contents of two memory locations directly.
- Memory location can be specified as the source or destination.
- Segment registers can not be used.

XCHG reg , mem

Machine code format

Example

XCHG AX,BX

Before execution AX = 0001H, BX = 0002H

After execution AX = 0002H, BX = 0001H

XCHG AX,[BX]

Before execution AX = 0001H

After execution AX = ?

XLAT

- XLAT (translate) replaces a byte in the AL register with a byte from a 256 byte, user-coded translation table.

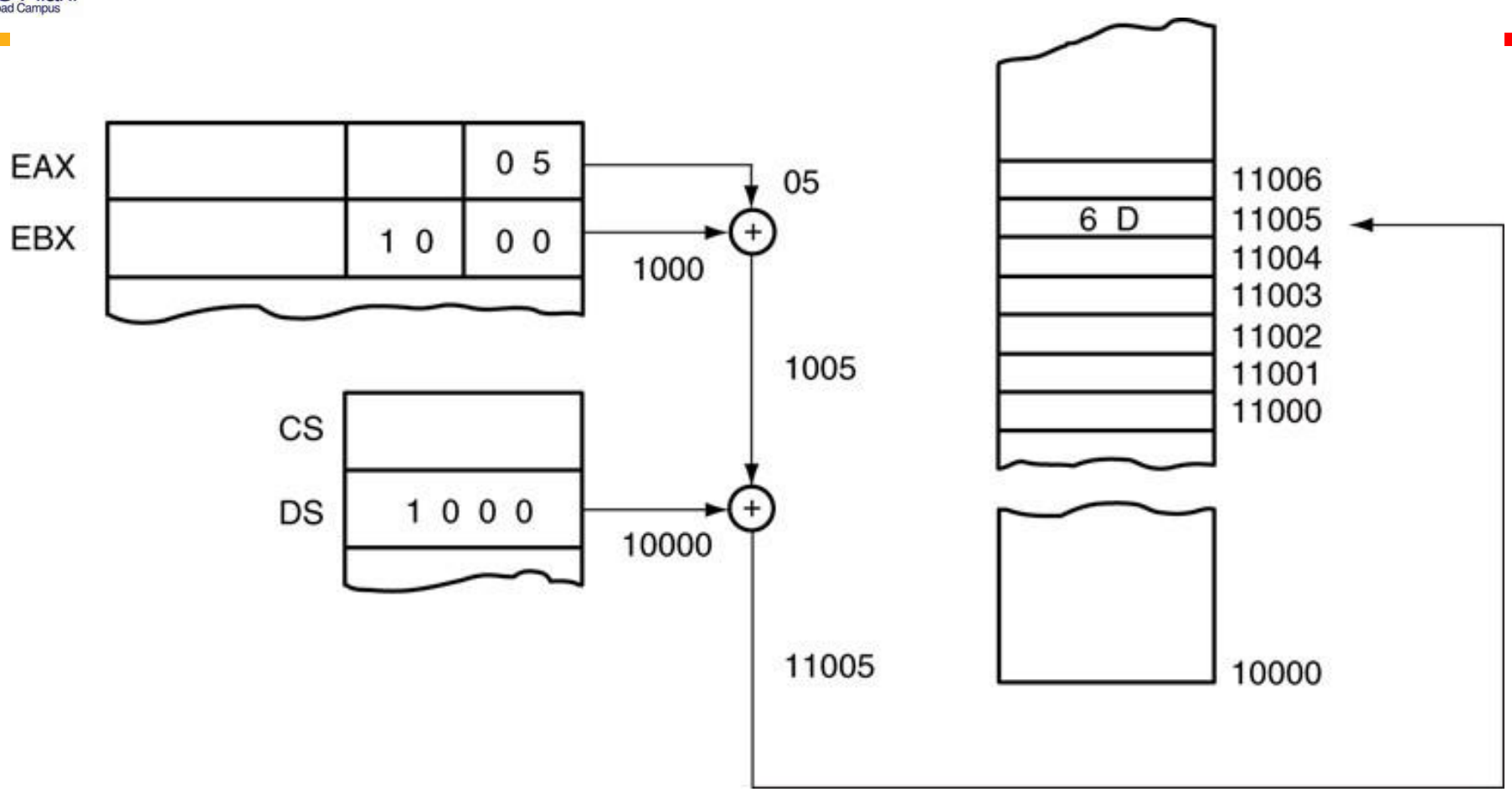
- Register BX is assumed to be pointed to the beginning of the table (i.e. beginning location of the table)

- Byte AL is used index in to the table (AL = 0).
- AL is replaced with byte at location [BX]+AL.

XLAT

1 1 0 1 0 1 1 1

Machine code format


EAX
0 5
EBX
1 0
0 0
05
1000
1005
CS
DS
1 0 0 0
10000
11005
6 D
11006
11005
11004
11003
11002
11001
11000
10000

Example(contd)….

DS = 2000
Assume BX is pointing beginning location of the table (i.e. BX = 0000 H).

```
MOV AL, 05
XLAT
```

After execution AL is copied with byte located by BX + AL = 0000 H + 05 H = 0005 H.
PA = DS*10 H+0005 H=20005 H
(i.e. content at location 20005 H is copied into AL = 6D H)

```
MOV AL, 07
XLAT
```

After execution what is the value of AL = ?

20000 H	00	3FH
20001 H	01	06 H
20002 H	02	5B H
20003 H	03	4F H
20004 H	04	66 H
20005 H	05	6D H
20006 H	06	7D H
20007 H	07	27 H
20008 H	08	7F H
20009 H	09	6F H

STACK

- Temporary memory (stack).
- It is used for storing data temporarily.
- The **stack** memory is special kind of memory which has the **Last In First Out** policy (**LIFO**) i.e. the data stored last will be the data which will retrieve first.

PUSH

- It is used for storing data into temporary memory (stack).
- Pushes the contents of the specified register / memory location on to the stack.
- The stack pointer is decremented by 2 , after each execution.
- The source of the word can be a general purpose register, a segment register, or memory.
- The SS and SP must be initialized before this instruction.
- No flags are affected.

PUSH

- Store data into LIFO stack memory
- 2/4 bytes involved
- Whenever 16-bit data pushed into stack
- MSB moves into memory [SP-1]
- LSB moves into memory [SP-2]
- Contents of SP register decremented by 2

The effect of the PUSH AX instruction on ESP and stack memory locations 37FFH and 37FEH. This instruction is shown at the point after execution.

PUSH

Push data from

- Registers/Segment Register
- Memory
- Flag Register

PUSH

- Always transfers 2 bytes of data to the stack;
  - 80386 and above transfer 2 or 4 bytes

- PUSHA instruction copies contents of the internal register set, except the segment registers, to the stack.

- PUSHA (**push all**) instruction copies the registers to the stack in the following order: AX, CX, DX, BX, SP, BP, SI, and DI.

- PUSHA requires 16 bytes of stack memory space to store all eight 16-bit registers
- the contents of the SP register are decremented by 16.

- PUSHAD instruction places 32-bit register set on the stack in 80386 - Core2.
  - PUSHAD requires 32 bytes of stack storage

PUSHF (**push flags**) instruction copies the contents of the flag register to the stack.

PUSHF

- Pushes flag register on to the stack; First the upper byte and then the lower byte is pushed on to it. The SP (stack pointer) is decremented by 2 for each PUSH operation.

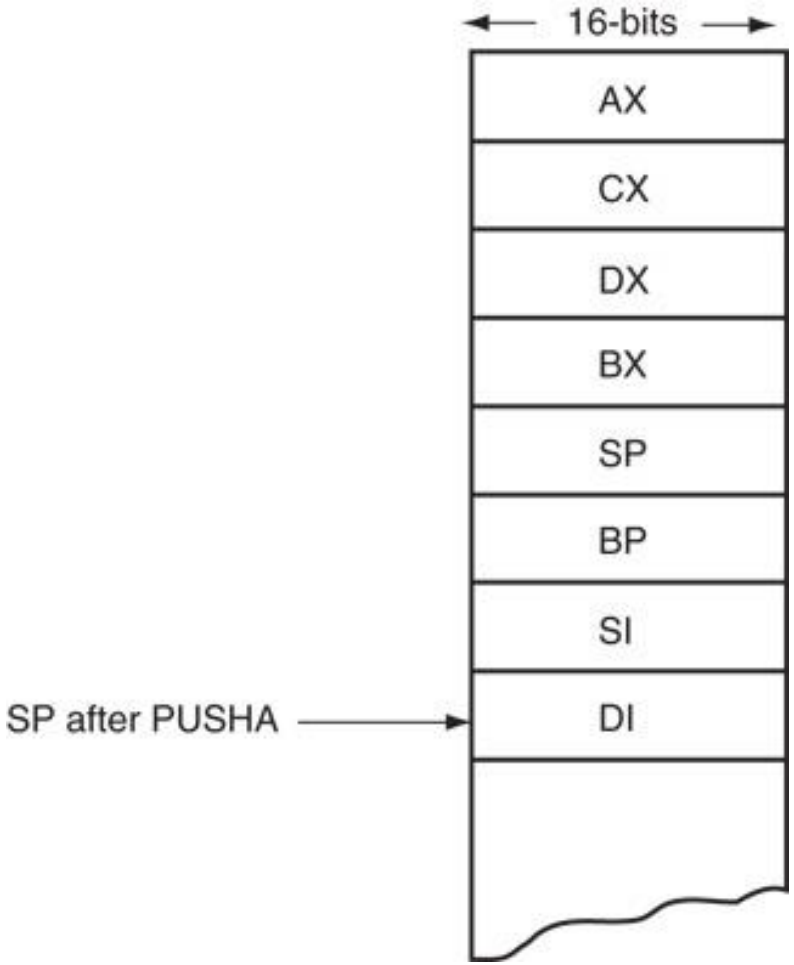
- The FLAGS themselves are not affected.

PUSHF

1 0 0 1 1 1 0 0

Machine code format

PUSHA instruction, showing the location and order of stack data.



PUSH

Example-PUSH operation

PUSH AX
PUSH EBX
PUSH DS
PUSH WORD PTR[BX]
PUSHF
PUSHFD
PUSHA
PUSHAD
PUSH 16-imm
PUSHD 32-imm

PUSH

Example-

PUSH AX
PUSH BX
PUSH SI
PUSH WORD PTR[BX]
PUSHF

70050	
$7004F_H$	AH
$7004E_H$	AL
$7004D_H$	BH
$7004C_H$	BL
$7004B_H$	$SI_{(High)}$
$7004A_H$	$SI_{(Low)}$
70049_H	$Mem_{(high)}$
70048_H	$Mem_{(low)}$
70047_H	$FLR_{(high)}$
70046_H	$FLR_{(low)}$

$SP \leftarrow SP-2$ $[004E_H]$
$7004E_H \leftarrow AX$
$SP \leftarrow SP-2$ $[004C_H]$
$7004C_H \leftarrow BX$
$SP \leftarrow SP-2$ $[004A_H]$
$7004A_H \leftarrow SI$
$SP \leftarrow SP-2$ $[0048_H]$
$70048_H \leftarrow MEM$
$SP \leftarrow SP-2$ $[0046_H]$
$70046_H \leftarrow FLAGS$

$SP:0050_H$ $SS:7000_H$

POP

- Performs the inverse operation of PUSH

- Ex: POP CX

  - not available as an immediate POP

POP

- POPA removes 16 bytes of data from the stack and places them into the following registers, in the order shown: DI, SI, BP, SP, BX, DX, CX, and AX.
- reverse order from placement on the stack by PUSHA instruction, causing the same data to return to the same registers

POPF

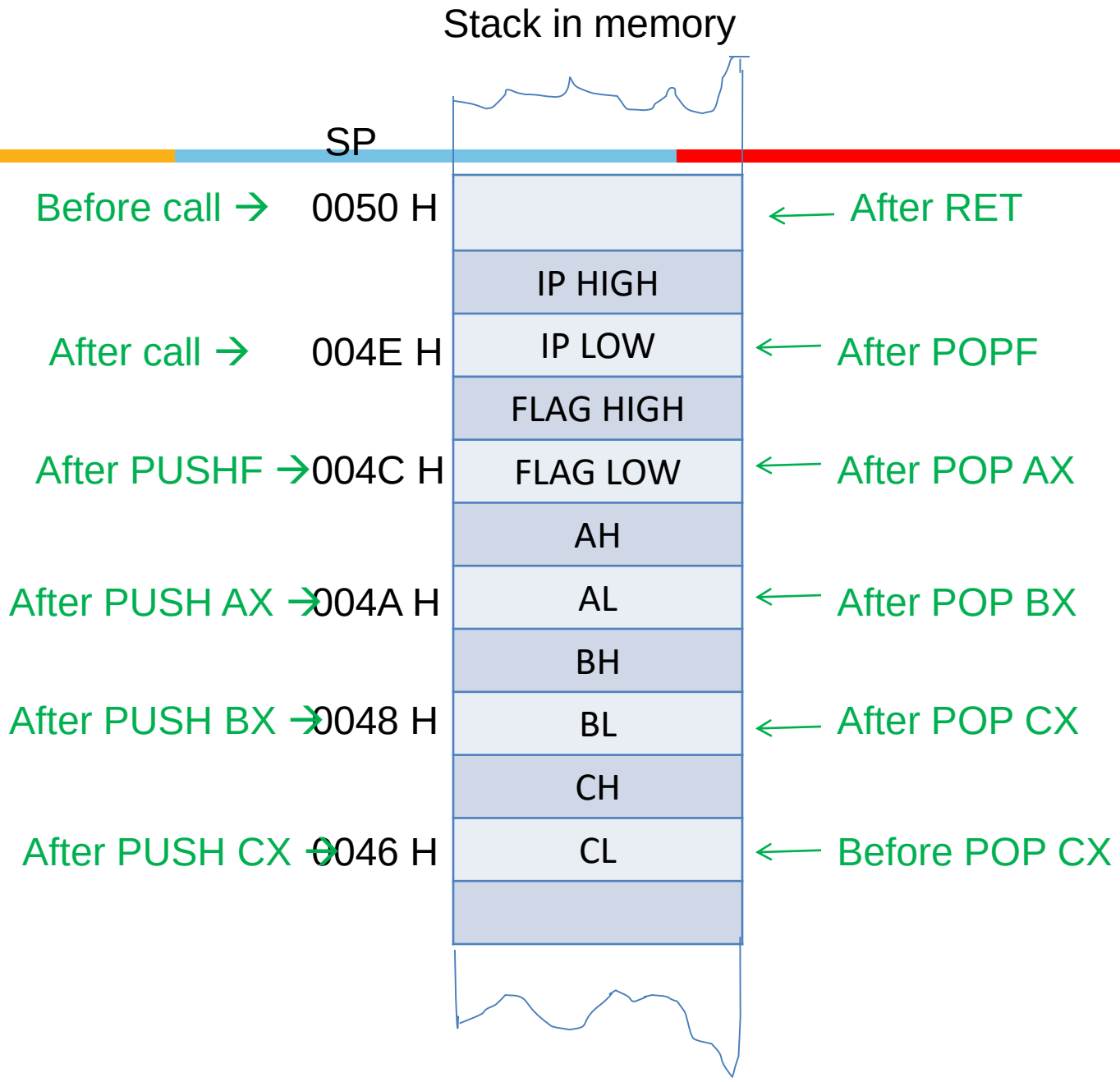
- Transfers a word from the current top of the stack to the FLAG register and increments the SP by 2.
- All the flags will be affected.
- PUSHF and POPF are used when there is a subprogram.

POPF

1 0 0 1 0 0 0 0

Machine code format

```
MUL TO PROC
NEAR
        PUSHF
        PUSH AX
        PUSH BX
        PUSH CX
          .
          .
          .
        POP CX
        POP BX
        POP AX
        POPF
        RET
MUL TO ENDP
```



Input / Output

- IN : Input byte or word
- OUT : Output byte or word

IN and OUT

- IN & OUT instructions perform I/O operations.
  - an IN instruction transfers data from an external Input device into AL, AX, or EAX
  - an OUT transfers data from AL, AX, or EAX to an external O/P device

IN

- IN instruction has two formats:
  - Fixed port: port number is specified directly in the instruction (port no: 0-255).
  - Variable port: port number is loaded into the DX register before IN instruction (port no : 0 – 65535).

IN acc , port no#

IN acc , DX

Machine code formats

OUT

- OUT transfers a byte or a word from AL register or AX register respectively, to an output port.
- OUT instruction has two formats:
  - Fixed port: port number is specified directly in the instruction (port no: 0-255).
  - Variable port: port number is loaded into the DX register before OUT instruction (port no : 0 – 65535).

OUT port no# , acc

1 1 1 0 0 1 0 w	port no #

OUT DX, acc

1 1 1 0 1 1 1 w

Machine code formats

Address Object data transfer

- LEA : Load effective address
- LDS : Load pointer using DS
- LES : Load pointer using ES
- LFS : Load pointer using FS
- LGS : Load pointer using GS
- LSS : Load pointer using SS

LEA(Load Effective Address)

- LEA transfers the offset of the source operand to a destination operand.
- The source operand must be a memory operand.
- The destination operand must be a 16-bit general purpose register.
- Does not effects flags.

LEA reg , mem

1 0 0 0 1 1 0 1	mod reg r/m

Machine code format

Example

LEA BX , [1234H]

Before execution BX = xxxxh

After execution BX =?

MOV BX , [1234H]

Before execution BX = xxxxh

After execution BX = ?

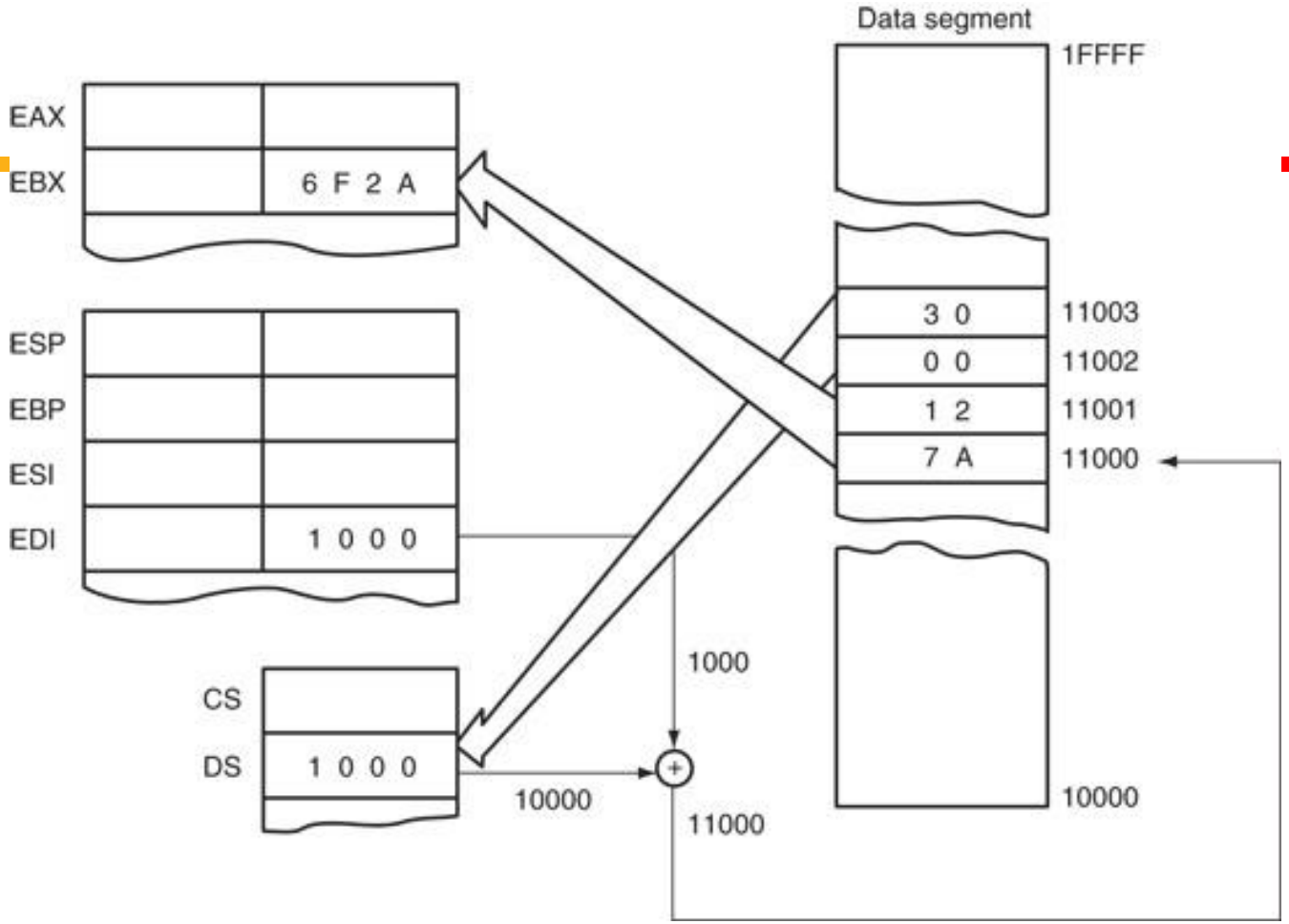
LDS (load pointer using DS)

- LDS transfers 32-bit pointer variable from source operand to destination operand and DS register.

LDS R, M

- The source operand must be a memory operand.
- The destination operand may be any 16-bit general purpose register.

- The first word of the pointer variable is transferred into 16-bit general purpose register.
- The second word of the pointer variable transferred into DS.



The LDS BX,[DI] instruction loads register BX from addresses 11000H and 11001H and register DS from locations 11002H and 11003H.

Accessing array in data segment

```
A       DW      0000, 1000
```

1000:0000 =	A[0]	04
1000:0001 =	A[1]	03
1000:0002 =	A[2]	02
1000:0003 =	A[3]	01
1000:0004 =	A[4]	00

```
MOV     SI , WORD PTR A
MOV     AX , WORD PTR A+2
MOV     DS , AX
```

With MOV instruction

After execution of all three MOV instructions
DS =0102
SI = 0304

```
LDS     SI , A
```

With LDS instruction

After execution of LDS
DS =0102
SI = 0304

LES (load pointer using ES)

- LES transfers 32-bit pointer variable from source operand to destination operand and ES.
- LES R, M
- The source operand must be a memory operand.
- The destination operand may be any 16-bit general purpose register.

- The first word of the pointer variable is transferred into 16-bit general purpose register.
- The second word of the pointer variable transferred into ES.

Accessing array in extra segment

```
B       DW      0000, 2000
```

2000:0000 = B[0]	04
2000:0001 = B[1]	03
2000:0002 = B[2]	02
2000:0003 = B[3]	01
2000:0004 = B[4]	00

```
MOV     DI , WORD PTR B
MOV     AX , WORD PTR B + 2
MOV     ES , AX
```

With MOV instruction

After execution of all three MOV instructions
ES = 0102
DI = 0000

```
LES     DI , B
```

With LES instruction

After execution of LES
ES = 0102
DI = 0304

Flag Register Data transfer

- LAHF : Load AH register from flags
- SAHF : Store AH register in flags
- PUSHF : Push flags onto stack
- POPF : Pops flags off stack

LAHF

- LAHF instruction transfers the rightmost 8 bits of the flag register into the AH register.
- Copies SF, ZF, AF, PF and CF into bits 7 ,6 ,4 ,2 and 0, respectively of AH.
- Contents of 5 ,3 ,1 are undefined.

Flag Register

LAHF

10011111

Machine code format

SAHF

- SAHF instruction transfers the AH register into the rightmost 8 bits of the flag register.
- Transfers bits 7 ,6 ,4 ,2 and 0 of AH register to SF, ZF, AF, PF and CF of FLAG register respectively.
- OF, DF, IF and TF are not affected.

Flag Register

SAHF

10011110

Machine code format

Additional Data Transfer Instructions (X386 onwards)

- MOVSX DST, SRC

  Ex: MOVSX CX, BL

- MOVZX DST, SRC

  Ex: MOVZX CX, BL

- BSWAP REG 32

  Ex: BSWAP EAX

Additional Data Transfer Instructions (X386 onwards)

- **MOVSX DST, SRC**

  Ex: MOVSX CX, BL

- SX– Sign extension
- Destination size > Source size

Example: MOVSX CX, BL

Assume BL= 80H

After execution of MOVSX instruction

BL=80H

CX= CH CL

CL=80H = 1000 0000

CH= 1111 1111= FFH

Thus **CX= FF80H**

Additional Data Transfer Instructions (X386 onwards)

- **MOVZX DST, SRC**

  Ex: MOVZX CX, BL

- ZX– Zero extension
- Destination size > Source size

Example: MOVZX CX, BL

Assume BL= 80H

After execution of MOVZX instruction

BL=80H

CX= CH CL

CL=80H = 1000 0000

CH= 0000 00000=00H

Thus **CX= 0080H**

Additional Data Transfer Instructions (X386 onwards)

- **BSWAP REG 32**

  Ex: BSWAP ECX
- CONVERT LITTLE ENDIAN FORMAT TO BIG ENDIAN FORMAT
- Only 32 bit registers

Example: BSWAP ECX

Assume ECX= **24 56 89 A0H**

After execution of BSWAP ECX instruction

ECX= A0 89 56 24H

STRING DATA TRANSFERS

- 80x86 is equipped with special instructions to handle string operations
- String: A series of data words (or bytes) that reside in consecutive memory locations

- Each allows data transfers as a single byte, word, or double word.

STRING DATA TRANSFERS

- Five string data transfer instructions:

  MOVS, LODS, STOS, INS, and OUTS.

- Before the string instructions are presented, the operation of the D flag-bit (direction), DI, and SI must be understood as they apply to the string instructions.

The Direction Flag

- The direction flag (D, located in the flag register) selects the auto-increment or the auto-decrement operation for the DI and SI registers during string operations.
  - used only with the string instructions

- The **CLD** instruction clears the D flag (D flag =0 or reset) and the **STD** instruction sets it (D flag =1 or set) .

  - **CLD** instruction selects the auto-increment mode and **STD** selects the auto-decrement mode

DI and SI

- During execution of string instruction, memory accesses occur through DI and SI registers.
  - DI offset address accesses data in the extra segment for all string instructions that use it
  - SI offset address accesses data by default in the data segment
  - Operating in 32-bit mode EDI and ESI registers are used in place of DI and SI.

DI and SI

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

Selects auto increment D=0
auto decrement D=1
operation for the DI & SI registers during string Ops
D is used only with strings

MOVS/MOVSB/MOVSW/MOVSD

- Copies a byte or word or double-word from a location in the data segment to a location in the extra segment
- Source –DS:SI
- Destination –ES:DI
- No Flags Affected
- For multiple-byte or multiple-word moves, the count to be in CX register
- Byte transfer, SI or DI increment or decrement by 1
- Word transfer, SI or DI increment or decrement by 2
- Double word transfer SI or DI increment or decrement by 4

MOVS/MOVSB/MOVSW/MOVSD

- 2nd method: Declaring the source and destination strings as DB → for byte transfer
- Declaring both as DW → for word transfer

MOVS with a REP

- The repeat prefix (REP) is added to any string data transfer instruction except LODS.
  - REP prefix causes CX to decrement by 1 each time the string instruction executes; after CX decrements, the string instruction repeats

- If CX reaches a value of 0, the instruction terminates and the program continues.

EX: If CX is loaded with 100 and a REP MOVSB instruction executes, the microprocessor automatically repeats the MOVSB 100 times.

COPY A BLOCK OF DATA FROM ONE MEMORY AREA TO ANOTHER MEMORY AREA-50 DATA

```
.data
Array1  db0ah,bch,deh,0f5h,11h, 56h,78h,0ffh,0ffh ,23h4ah, …
Array2   db   50 dup(0)
.code
startup
MOV     CX, 32H
LEA      SI, array1
LEA     DI, array2
CLD
REP   MOVSB
.EXIT
END
```

LODS/LODSB/LODSW

- Loads AL, AX with data at segment offset address indexed by the SI register.
- 1 is added to or subtracted from SI for a byte-sized LODS (LODSB)
- 2 is added or subtracted for a word-sized LODS (LODSW).

- 4 is added or subtracted for a doubleword-sized LODS.

LODS/LODSB/LODSW /LODSD

Loads AL or AX or EAX with the data stored at the data segment

•Offset address indexed by SI register

•After loading contents of SI INC if D = 0 & DEC if D = 1

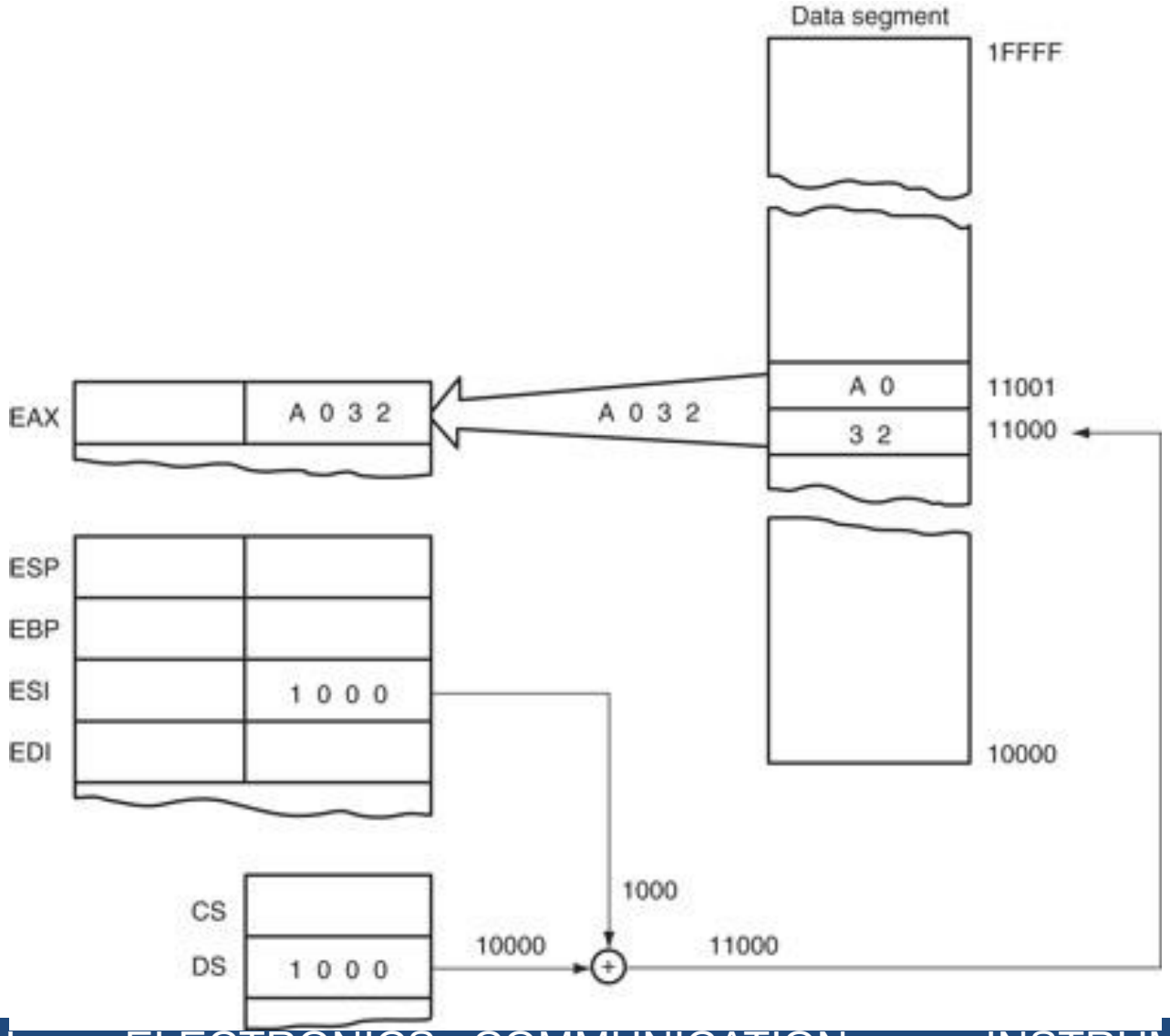
LODSB ; AL = DS:[SI]; SI = SI ±1

LODSW ; AX = DS:[SI]; SI = SI ±2

LODSD; EAX = DS:[SI]; SI = SI±4

•**LODS** affects no FLAGs

The operation of the LODSW instruction if DS=1000H, D=0,11000H,=32 11001H = A0. This instruction is shown after AX is loaded from memory, but before SI increments by 2.



BITS Pilani
Hyderabad Campus

Ex:

```
CLD                 ; Clears the direction flag
                    ; so SI is Automaticallyincremented.

MOV SI,OFFSET SOURCE-STRING
                    ; point SI at start of string

LODS SOURCE-STRING  ; Copy byte or word from string
                      to  AL or AX.
```

STOS /STOSB/STOSW

- Stores AL, AX into the Extra segment memory location addressed by the DI register.
- STOSB (**stores a byte**) stores the byte in AL at the extra segment memory location addressed by DI.
- STOSW (**stores a word**) stores AX in the memory location addressed by DI.
- After the byte (AL), word (AX), or doubleword (EAX) is stored, contents of DI increment or decrement.

STOS /STOSB/STOSW

Stores AL or AX or EAX into the Extra segment ES memory at Offset address indexed by DI register

•After storing contents in DI, INC if D = 0 & DEC if D = 1

STOSB ; ES:[DI]=AL; DI = DI ±1

STOSW ; ES:[DI]=AX; DI = DI ±2

STOSD; ES:[DI]=EAX; DI = DI ±4

STOS affects no FLAGs

Write an ALP to fill a set of 100 memory locations starting at displacement ‘DIS1’ with the value F6H

```
 .DATA
 DAT1          DB     100 DUP(?)
.CODE
.STARTUP
MOV  DI,  OFFSET DAT1
MOV AL, 0F6H
MOV CX, 64H
CLD
REP    STOSB
.EXIT
END
```

INS

- Transfers a byte or word of data from an I/O device into the extra segment memory location addressed by the DI register.
  - I/O address is contained in the DX register
- Useful for inputting a block of data from an external I/O device directly into the memory.

- Ex : One application transfers data from a disk drive to memory.
  - disk drives are often considered and interfaced as I/O devices in a computer system

THREE basic forms of the INS.

- INSB inputs data from an 8-bit I/O device and stores it in a memory location indexed by SI.
- INSW instruction inputs 16-bit I/O data and stores it in a word-sized memory location.
- INSD instruction inputs 32-bit I/O data and stores it in a word-sized memory location.
- These instructions can be repeated using the REP prefix
  - allows an entire block of input data to be stored in the memory from an I/O device

OUTS

- Transfers a byte or word data from the data segment memory location address indexed by SI to an I/O device.
  - I/O device addressed by the DX register as with the INS instruction