CSE-3103: Microprocessor and Microcontroller

Dept. of Computer Science and Engineering
University of Dhaka

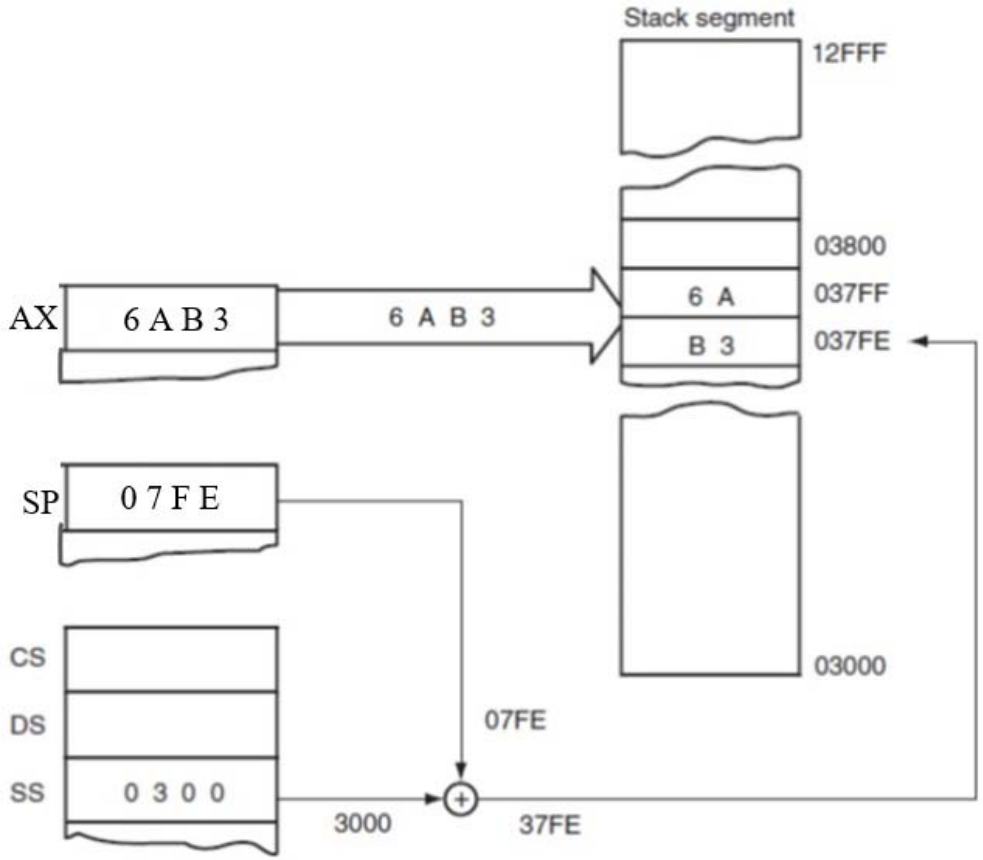
Prof. Sazzad M.S. Imran, PhD
Dept. of Electrical and Electronic Engineering
sazzadmsi.webnode.com

PUSH and POP Instructions

Store and retrieve data from LIFO stack memory.

6 forms of PUSH and POP instructions →

(1) Register addressing.
(2) Memory addressing.
(3) Immediate addressing.
(4) Segment register addressing.
(5) Flags.
(6) All registers.



PUSH →

transfers 2 bytes of data to stack.
source of data →

any internal 16-bit register,
immediate data,
any segment register,
any 2 bytes of memory data.

PUSH AX →

SS:[SP-1] = AH, SS:[SP-2] = AL, SP = SP-2.

PUSH and POP Instructions

PUSHA →

pushes all internal 16-bit registers onto stack.

exception →

segment registers, IP, flag register,

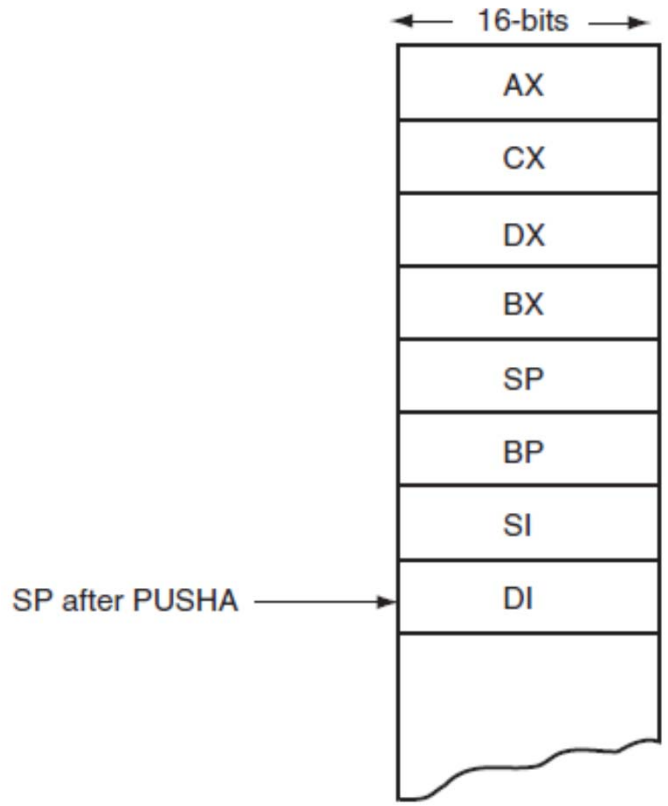
requires 16 bytes of stack memory space =

8 × 16-bit registers.

SP = SP-16.

PUSHF →

copies contents of flag register to stack.



PUSH and POP Instructions

POP →

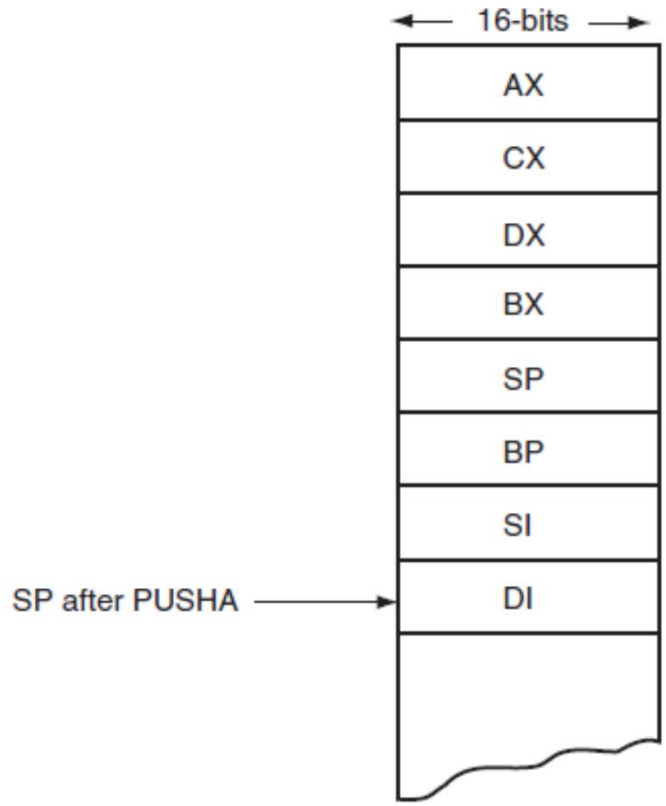
removes data from stack and places it into
- 16-bit register,
- segment register,
- 16-bit memory location.

POPF →

removes 16-bit number from stack,
places it into flag register.

POPA →

removes 16 bytes of data from stack,
places them into registers →
- DI, SI, BP, SP, BX, DX, CX, AX.



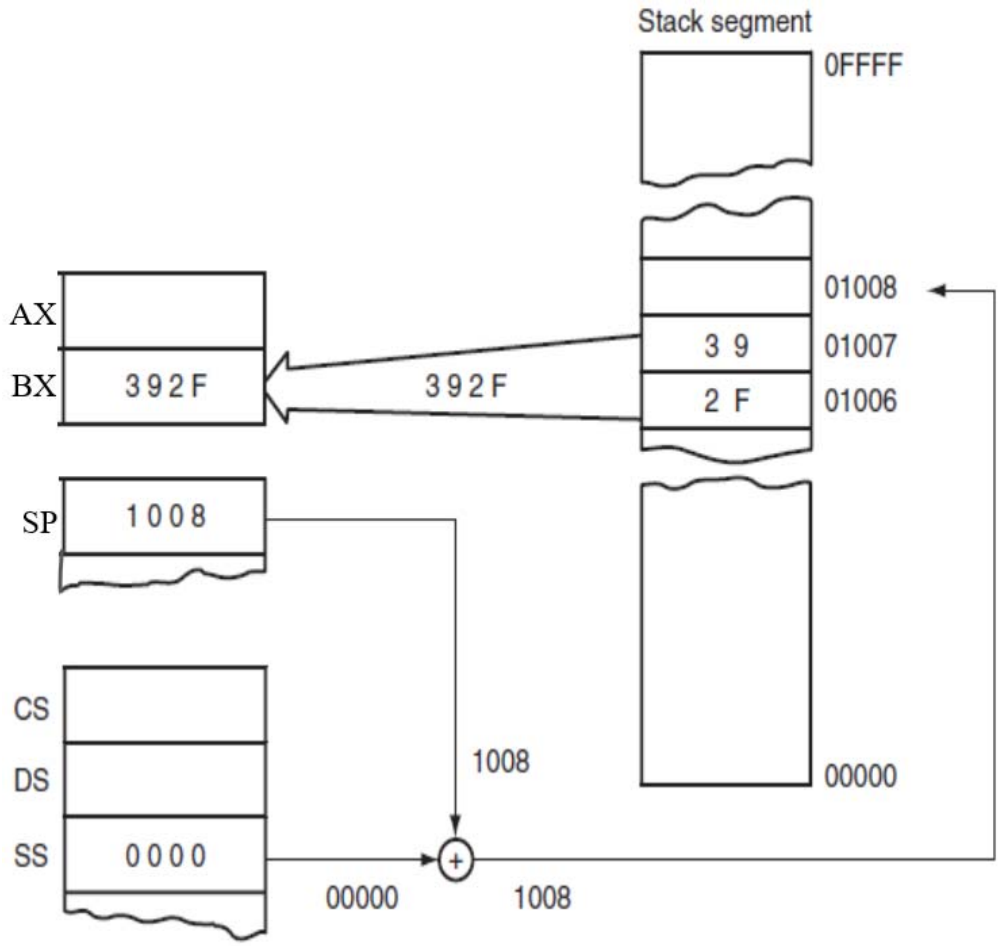
PUSH and POP Instructions

POP BX →

BL = SS:[SP],
BH = SS:[SP+1],
SP = SP+2.

PUSH AX →

SS:[SP-1] = AH,
SS:[SP-2] = AL,
SP = SP-2.



LOOP Instructions

LOOP = way to repeat block of statements specific number of times.
LOOP instruction = decrement of CX + JNZ conditional jump.
CX = used as default counter.

Each time loop repeats

CX = CX-1.
If CX != 0, it jumps to indicated label.
If CX = 0, next sequential instruction executes.

Format: LOOP Short-label

LOOP Instructions

```
_DATA   SEGMENT                          ; start data segment
        BLOCK1   DW   100   DUP(?)       ; 100 words for BLOCK1
        BLOCK2   DW   100   DUP(?)       ; 100 words for BLOCK2
_DATA   ENDS

_CODE   SEGMENT                          ; start code segment
STARTUP:                                 ; start program
        MOV     AX, DS
        MOV     ES, AX                   ; overlap DS and ES
        CLD                              ; select auto-increment
        MOV     CX, 100                  ; load counter
        MOV     SI, OFFSET  BLOCK1       ; address BLOCK1
        MOV     DI, OFFSET  BLOCK2       ; address BLOCK2
L1:     LODSW                            ; load AX with BLOCK1
        ADD     AX, ES:[DI]              ; add BLOCK2
        STOSW                            ; save answer to BLOCK2
        LOOP    L1                       ; repeat 100 times
_CODE   ENDS
END   STARTUP
```

LOOP Instructions

Conditional LOOPs →

LOOPE jumps if
CX != 0 and
equal condition exists.
It will exit loop if
condition is not equal or
CX register decrements to 0.

LOOPNE jumps if
CX != 0 and
not-equal condition exists.
It will exit loop if
condition is equal or
CX register decrements to 0.

Format-
LOOPE/LOOPZ Short-label
LOOPNE/LOOPNZ Short-label

String Instructions

String = series of data words/bytes,
reside in successive memory locations.

Characteristics of string instruction →

(1) Can move data from one block of memory locations to another one.
(2) String of data elements can be scanned for specific data value.
(3) Successive elements of two strings can be compared.

LODS →

AL or AX ← data stored at data segment addressed by SI.
After loading →

contents of SI increment, if DF = 0 or
decrement, if DF = 1.

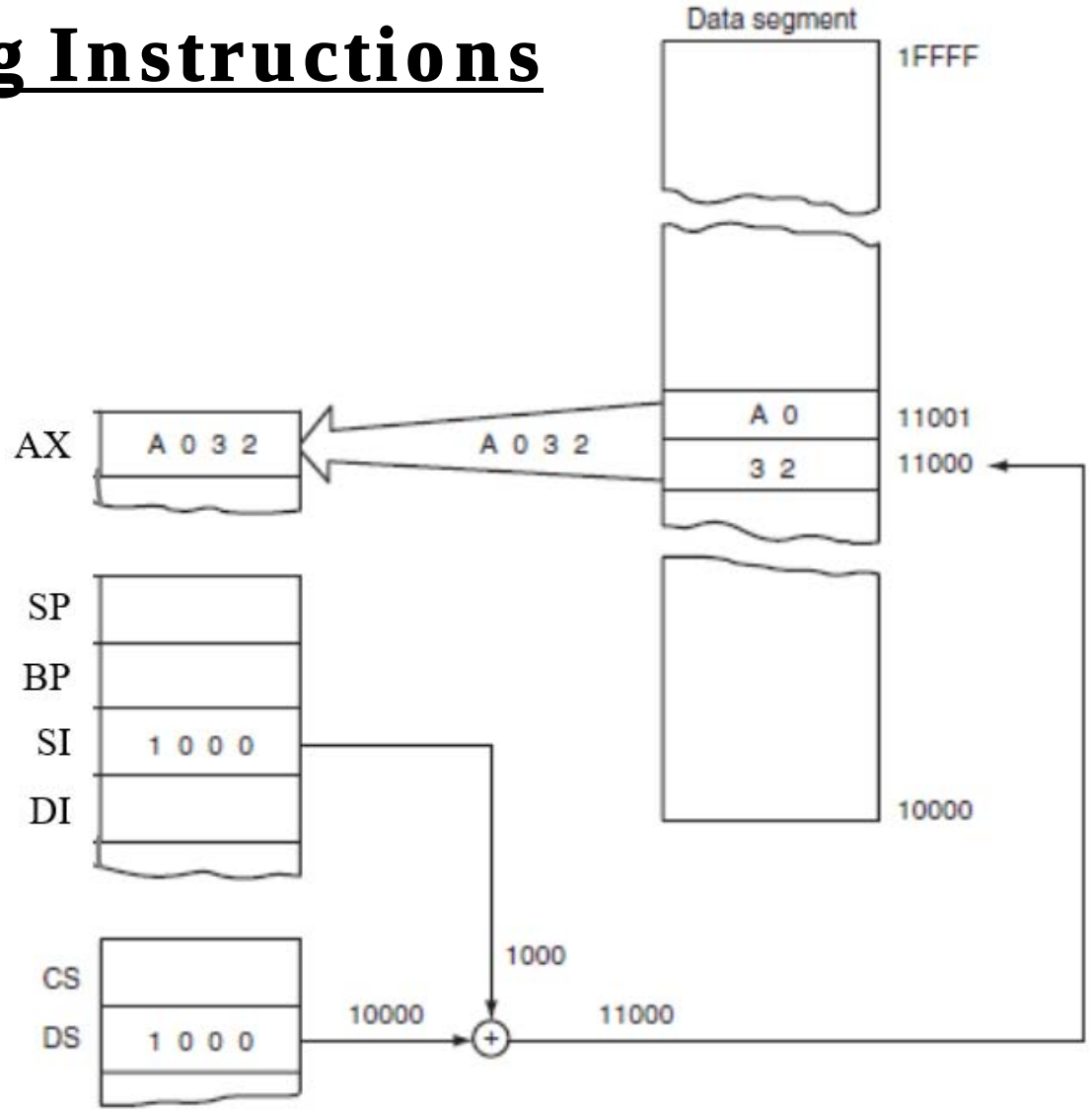
1 is added to or subtracted from SI for byte-sized LODS.
2 is added or subtracted for word-sized LODS.

LODSB = byte is loaded into AL.
LODSW = word is loaded into AX.

String Instructions

LODSW instruction →

DS = 1000H,
DF = 0,
11000H = 32,
11001H = A0.



String Instructions

STOS →

stores AL or AX at extra segment memory location addressed by DI.

STOSB →

stores byte in AL at extra segment memory location addressed by DI.

STOSW →

stores AX in extra segment memory location addressed by DI.

After byte, or word is stored →

DI increments or decrements by 1 or 2.

MOVS →

copy data item from source string to destination string.

Source string = DS:SI,

Destination string = ES:DI.

ES:DI ← DS:SI

MOVSB = move string byte,

MOVSW = move string word.

After data is moved →

both SI and DI are automatically incremented or decremented.

Index registers are incremented if DF = 0 and decremented if DF = 1.

Increment/decrement count is 1 for byte move and 2 for word move.

String Instructions

INS →

transfers byte, word, or double-word of data from I/O device,
into extra segment memory location addressed by DI.
I/O address = DX register.
ES:DI ← [DX]

INSB →

inputs data from 8-bit I/O device,
stores it in byte-sized memory location indexed by DI.

INSW →

inputs 16-bit I/O data,
stores it in word-sized memory location.

INSD →

inputs double-word/32-bit I/O data.

OUTS →

transfers byte, word, or double-word of data to I/O device,
from data segment memory location addressed by SI.
I/O device address = DX register.
[DX] ← DS:SI

String Instructions

SCAS →

searches particular character or set of characters in string.
Data item to be searched should be in
AL for SCASB,
AX for SCASW.
String to be searched should be in memory pointed by ES:DI.

REP →

repeats string operations required for processing arrays of data.
added to any string data transfer instruction, except LODS.
used with MOVS and STOS.

each time string instruction executes →
CX decrements by 1.
if CX ≠ 0 →
string instruction repeats.
if CX = 0 →
instruction terminates,
program continues with next sequential instruction.

String Instructions

REP →

CX = 100,

REP STOSB →

microprocessor automatically repeats STOSB 100 times.

REPE/REPZ →

repeat while not end of string and strings are equal.

REPNE/REPNZ →

repeat while not end of string and strings are not equal.

REPE and REPNE are used with CMPS and SCAS.

String Instructions

```
; example from “Conditional Jumps”
; SCASB searches a table of 100 bytes for 0AH
; address of TABLE is assumed to be in ES:DI
        MOV     CX, 100                 ; load counter
        MOV     AL, 0AH                 ; load AL with 0AH
        CLD                             ; auto-increment
        REPNE  SCASB                    ; search for 0AH
                                        ; repeat SCASB until CX=0
        JCXZ    NOT_FOUND               ; if not found
        STC                             ; set carry if found
NOT_FOUND
```