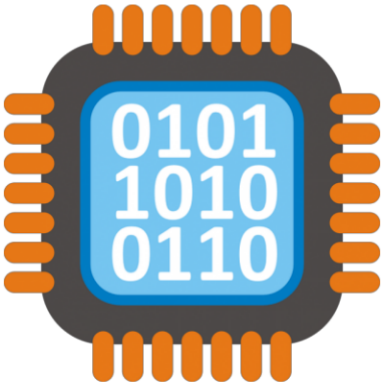
Secure Assembly Coding

Week # 11 Lectures



Dr. Qasem Abu Al-Haija,
Department of Cybersecurity,

String/Array Instructions

8086 Instruction Set

Group 7 : String Instructions

String is an array of data bytes/words, stored in a consecutive memory Locations.

MNEMONICS	FUNCTION
MOVSB	*Move string byte from DS:[SI] to ES:[DI]*
MOVSW	*Move string word from DS:[SI] to ES:[DI]*
CMPSB	*Compare string byte (Done by subtracting byte at ES:[DI] from the byte at DS:[SI]). Only flags are affected and the content of bytes compared is unaffected.*
CMPSW	*Compare string word (Done by subtracting word at ES:[DI] from the word at DS:[SI]). Only flags are affected and the content of words compared is unaffected.*
LODSB	*Load string byte at DS:[SI] into AL*
LODSW	*Load string word at DS:[SI] into AX*
STOSB	*Store string byte in AL at ES:[DI]*
STOSW	*Store string word in AX at ES:[DI]*
SCASB	*Compare string byte (Done by subtracting byte at ES:[DI] from the byte at AL). Only flags are affected and the content of bytes compared is unaffected.*
SCASW	*Compare string word (Done by subtracting word at ES:[DI] from the byte at AX). Only flags are affected and the content of words compared is unaffected.*
REP	*Decrement CX and Repeat the following string operation if CX ≠ 0.*
REPE or REPZ	*Decrement CX and Repeat the following string operation if CX ≠ 0 and ZF=1.*
REPNE or REPNZ	*Decrement CX and Repeat the following string operation if CX ≠ 0 and ZF=0.*

8086 Instruction Set

Group 7 : String Instructions

EX: MOVS WORD.

Assume that:

(DF)=0 ***(DS)=1000_{16}*** ***(SI)=0002_{16}*** ***(ES)=3000_{16}***

(DI)= 0004_{16} ***(10002)=1234_{16}*** ***(30021)= 05I6***

Then, after this MOVS:

(30004)=1234_{16}, ***(SI)=0004_{16},*** ***(DI)=0006_{16}***

Assuming (10002_{16}) = 1234_{16} ➔ 8086 Insts to accomplish this???

8086 Instruction Set

Group 7 : String Instructions

```
CLD                          ;DF = 0
MOV     AX, 1000H            ;DS = 1000_16
MOV     DS, AX
MOV     BX, 3000H            ;ES = 3000_16
MOV     ES, BX
MOV     SI, 0002H            ;Initialize SI to 0002_16
MOV     DI, 0004H            ;Initialize DI to 0004_16
MOVSW
```

8086 Instruction Set

Group 7 : String Instructions

EX: if (DF) = 0, (DS) = 1000_{16}, (ES) = 3000_{16}, (SI) = 0002_{16},

(DI) = 0004_{16}, (10002) = 1234_{16}, (30004) = 1234_{16}

Then, after ***CMPS WORD:***

(10002) = 1234_{16}, (30004) = 1234_{16}, (SI) = 0004_{16}, (DI) = 0006_{16}.

Flags: CF = 0, PF= 1, AF= 1, ZF= 1, SF=0, OF =0

EX: if : (DI) = 0000_{16} , (ES) = 2000_{16} , (DF) = 0, (20000) = 05_{16}, (AL) = 03_{16},

Then, after SCASB:

DI will contain 0001_{16} because (DF) = 0.

All flags are affected based on the operation (AL) - (20000).

8086 Instruction Set

Group 7 : String Instructions

Example:

```
MOV AX,5000 H
MOV DS,AX
MOV SI,2500H
MOV AX,6000H
MOV ES,AX
MOV DI,2500H
CLD
MOV CX,0004H
REPNE CMPSB
```

DS:[SI] - ES:[DI] and increment DI and SI (REPNE)

MEMORY SOURCE	CONTENT	MEMORY DEST	CONTENT
52500	42	62500	98
52501	45	62501	34
52502	34	62502	34
52503	23	62503	23

(CX=0004h)

(DS=5000h)(SI=2500h)(DI=2500h)(ES=6000h)

- 1st time REPNE CMPSB
  DS:[SI] - ES:[DI] i.e. [52500]-[62500] i.e. 42-98(NE)
  SI =2501 and DI = 2501
  CX=0003h(this is cos of REP prefix)
- 2ND time REP CMPSB

DS:[SI] - ES:[DI] i.e. [52501]-[62501] i.e. 45-34(NE)
SI =2502 and DI = 2502
CX=0002h(this is cos of REP prefix)

- *3rd Time* REPNE CMPSB

DS:[SI] - ES:[DI] i.e. [52502]-[62502] i.e. 34-34(E)
SI =2503 and DI = 2503
CX=0001h(this is cos of REP prefix)

- *4TH Time* REPNE CMPSB will not happen cos locations are no longer NE

Example:

LODSB	**MA = (DS) x 16_{10} + (SI)** **(AL) ← (MA)** **If DF = 0, then (SI) ← (SI) + 1** **If DF = 1, then (SI) ← (SI) – 1**
LODSW	**MA = (DS) x 16_{10} + (SI)** **(AX) ← (MA ; MA + 1)** **If DF = 0, then (SI) ← (SI) + 2** **If DF = 1, then (SI) ← (SI) – 2**

Emulator Example

```
data segment

   STR1 db 1, 2, 3, 4, 5

   STR2 db 6, 0, 7, 8, 9

   pkey db "press any key...$"

ends

stack segment
   dw   128  dup(0)
ends
```

```
code segment
start: ; set segment registers:
    mov ax, data
    mov ds, ax
    mov si, offset STR2
    cld

    mov cx, 5
    rep    movsb

    lea dx, pkey
    mov ah, 9
    int 21h        ; output string at ds:dx

    ; wait for any key....
    mov ah, 1
    int 21h

    mov ax, 4c00h ; exit to operating system.
    int 21h
ends

end start ; set entry point and stop the assembler.
```

Flags Instructions

8086 Instruction Set

Group 6 : Flags manipulation Instructions

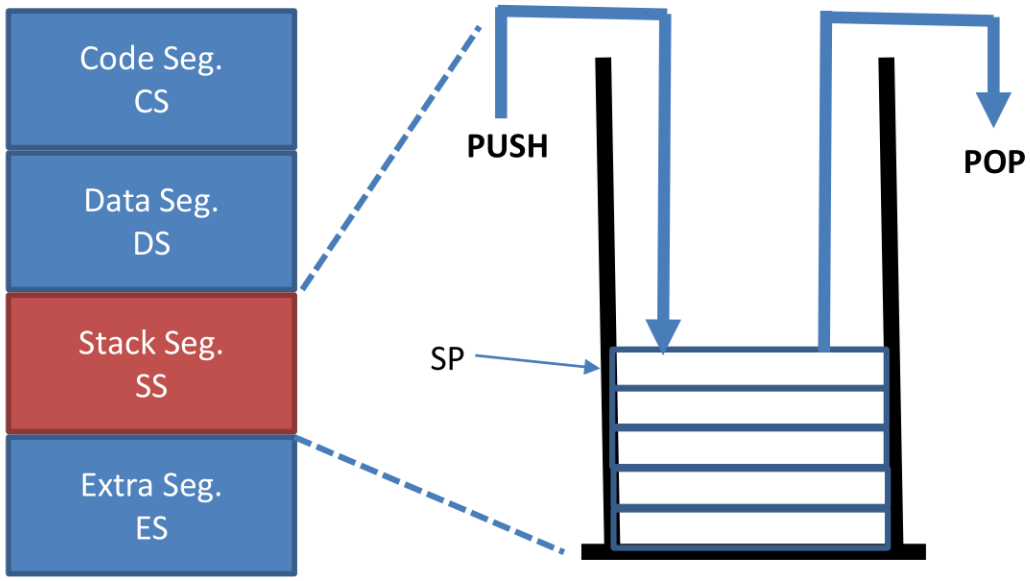
- These instructions are ZERO operand instructions.*(Implied addressing Modes)*
- Can be executed any where in the code.

Mnemonics	Function
LAHF	Load low byte of flag register into AH
SAHF	Store AH into the low byte of flag register
PUSHF	Push flag register's content into stack
POPF	Pop top word of stack into flag register
CMC	Complement carry flag (CF = complement of CF)
CLC	Clear carry flag (CF= 0)
STC	Set carry flag (CF= 1)
CLD	Clear direction flag (DF= 0)
STD	Set direction flag (DF= 1)
CLI	Clear interrupt flag (IF= 0)
STI	Set interrupt flag (IF=1)

STACK

The STACK

- Consists of 16-bit locations
- SP points at the top of the stack
- PUSH instruction is used add data to the top of the stack
- POP instruction is used to remove items from the top of the stack
- STACK is a LIFO structure
- SP is decremented with every PUSH
- SP is incremented with every POP
- Procedures use the stack
- The CALL instruction adds the return address on the top of the stack
- The RET instructions removes the address from the top of the stack
- Every PUSH Instruction must be associated with a matching POP
- Every CALL must also be associated with a matching RET
- Must be careful when using PUSH, POP, CALL, and RET

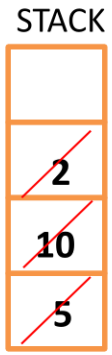


Using the STACK

- Useful for storing and retrieving data (16-bit)
- Useful for implementing the concept of local variables
- Can be used to reverse the order of stored data
- Useful for problems requiring backtracking

```
MOV   AX,5
MOV   BX,10
MOV   CX,2
PUSH  AX
PUSH  BX
PUSH  CX
DEC   AX
ADD   BX,2
MOV   CX, BX
POP   CX
POP   BX
POP   AX
```

AX	BX	CX
5	10	2
4	12	12
5	10	2

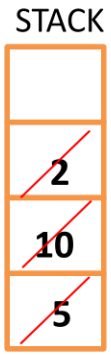


Note: the POPs are in reverse order of the PUSHs

```
MOV   AX,5
MOV   BX,10
MOV   CX,2
PUSH  AX
PUSH  BX
PUSH  CX
DEC   AX
ADD   BX,2
MOV   CX, BX
POP   AX
POP   BX
POP   CX
```

Note: the POPs are in same order of the PUSHs

AX	BX	CX
5	10	2
4	12	12
2	10	5



Note: register values are reversed

Call pushes the address of the next instruction on the stack.
RET: returns back the address from the top of the stack

REMEMBER

EXAMPLE: USING CALL, RET, PUSH AND POP

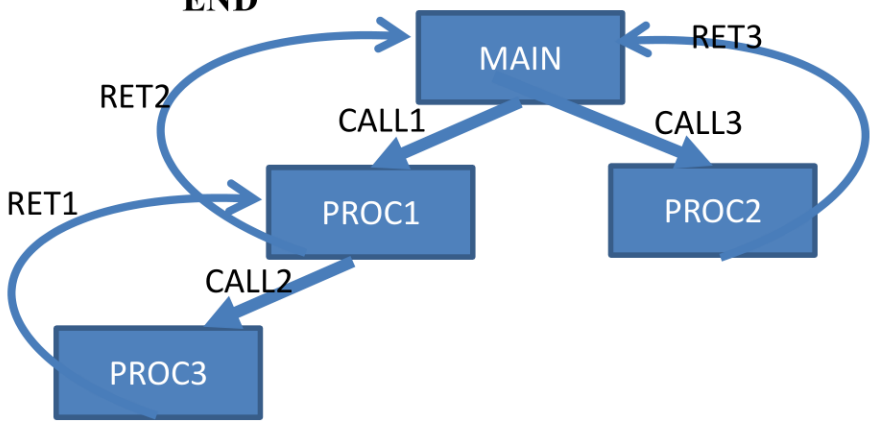
```
.DATA
MPROC1 DB "IN PROCEDURE 1",0DH,0AH,"$"
MPROC2 DB "IN PROCEDURE 2",0DH,0AH,"$"
MPROC3 DB "IN PROCEDURE 3",0DH,0AH,"$"

.CODE
  MOV    AX,5
  PUSH    AX
  CALL    PROC1
  CALL   PROC2
  MOV    AH,4CH
  INT      21H

  PROC1  :
        MOV AH,9
        LEA DX, MPPOC1
        INT 21H
        MOV   AX,2
        PUSH   AX
        POP    AX
        CALL  PROC3
  RET
```

```
PROC2 :
        MOV AH,9
        LEA DX, MPROC2
        INT 21H
    RET

PROC3 :
        MOV AH,9
        LEA DX, MPROC3
        INT 21H
  RET
END
```



EXAMPLE: USING CALL, RET, PUSH AND POP

Based on the Emulator:
CS: 0725
SS: 0715

```
EXAMPLE: USING CALL, RET, PUSH AND POP

.DATA
MPROC1 DB "IN PROCEDURE 1",0DH,0AH,"$"
MPROC2 DB "IN PROCEDURE 2",0DH,0AH,"$"
MPROC3 DB "IN PROCEDURE 3",0DH,0AH,"$"

.CODE
  MOV    AX,5
  PUSH   AX
  CALL   PROC1
  CALL   PROC2
  MOV    AH,4CH
  INT    21H

 PROC1  :
        MOV AH,9
        LEA DX, MPPOC1
        INT 21H
        MOV    AX,2
        PUSH   AX
        POP    AX
        CALL  PROC3
        MOV CX, 1
 RET
```

Address: 000E
Address: 0011
Address: 0015
Address: 0024
Return to Address: 000E

Address:0028
Return to Address:0011

```
PROC2 :
        MOV AH,9
        LEA DX, MPROC2
        INT 21H
  RET
```

Address:0030
Return to Address:0024

```
PROC3 :
        MOV AH,9
        LEA DX, MPROC3
        INT 21H
 RET
END
```

8086 Assembler- Examples

Writing 8086 program using line assembler

Example: Write a program to add a word type data located at offset 0800H (Least Significant Byte) and 0801H (Most Significant Byte) in the segment address 3000H to another word type data located at offset 0700H (Least Significant Byte) and 0701H (Most Significant Byte) in the same segment. Store the result at offset 0900H and 0901H in the same segment. Store the carry generated in the above addition in the same segment at offset 0902H.

```
        MOV AX, 3000H
        MOV DS, AX              ; initialize DS with value 3000H
        MOV AX, [800H]          ; get first data word in AX
        ADD AX, [700H]          ; add AX with second data word
        MOV [900H], AX          ; store AX at the offset 900H & 901H
        JC   CARRY              ; if carry=1, go to the place CARRY
        MOV [902H], 00H         ; no carry; hence store 00H at the offset 902H
        JMP END                 ; go to the place END
CARRY:  MOV [902H], 01H         ; store 01H at the offset 902H
END:    HLT                     ; stop
```

Writing 8086 program using line assembler

Example: Write a program to find the smallest word in an array of 100 words stored sequentially in the memory starting from the offset 1000H in the segment address 5000H and store the result at the offset 2000H in the same segment.

```
        MOV CX, 99          ; initialize CX with the number of comparisons (=100-1)
        MOV AX, 5000H
        MOV DS, AX          ; initialize DS with the segment address 5000H
        MOV SI, 1000H       ; initialize SI with offset 1000H
        MOV AX, [SI]        ; get the first word in AX
START:  INC SI
        INC SI              ; increment SI twice to point the next word
        CMP AX, [SI]        ; compare next word with the word in AX
        JC REPEAT           ; if AX is smaller then go to REPEAT
        MOV AX, [SI]        ; replace the word in AX with the smaller word
REPEAT: LOOP   START        ; repeat the operation from START
        MOV  [2000H], AX    ; store smallest number in AX at the offset 2000H
        HLT                 ; stop
```

Writing 8086 program using line assembler

Example: Write a program to convert the 8-bit binary number FFH into BCD number. The result is to be stored at memory locations 3000H: 2000H and 3000H: 20001H.

```
MOV AX, 00FFh        ; Move the data FFh in AX with upper byte as 00H
MOV BL, 100          ; Store 100 decimal (or 64H) in BL
DIV BL               ; Divide AX by BL to find number of hundreds in the binary number
MOV CL, AL           ; Move the quotient in AL (no. of hundreds) into CL
MOV AL, AH           ; Move the remainder in AH into AL
MOV AH, 00           ; Clear AH
MOV BL, 10           ; Store 10 decimal (or 0AH) in BL
DIV BL               ; Divide AX by BL to find number of tens in the binary number.
                     ; AH has the remainder, which is no. of ones in the binary number.
OR AL, AH            ; Perform OR between AL & AH to concatenate number of tens & ones.
MOV BX, 3000H
MOV DS, BX           ; Initialize DS with 3000H
MOV [2000H], CL      ; Move the value of CL to memory
MOV [2001H], AL      ; Move the value of AL to memory
HLT                  ; Stop
```

Writing 8086 program using line assembler

Example: Write a program to add a byte-type data located at the offset address 0800H in the segment address 3000H to another byte-type data located at the offset address 0700H in the same segment. Store the result and the Carrey generated in the offset addresses 0900H and 0901H in the same segment, respectively and in the stack segment. (Firstly, load the assigned locations with the values F3H and ECH respectively).

Program:

```
        MOV AX, 3000H     ; Load the value 3000H in AX.
        MOV DS, AX        ; Initialize DS with the value 3000H.
        MOV AL, [800H]    ; Move the first data byte to AL.
        ADD AL, [700H]    ; Add AL with the second data byte.
        MOV [900H], AL    ; Store AL at the offset address 900H.
        JC CARRY          ; If carry = 1, jump to CARRY.
        MOV [901H], 00H   ; If there is no carry, store 00H.
        JMP END           ; Jump to END.
CARRY:  MOV [901H], 01H   ; Store 01H at the offset address 901H.
END:    HLT               ; Terminate program execution.
```

Note: Instead of the AL register, any other 8-bit general purpose register (BL, BH, CL, etc.) can be used in the program.

Writing 8086 program using line assembler

Example: Write an assembly language program to: Move one hundred word-type data from offset address 1000H in the segment 5000H using MOVSW instruction. (Firstly load the assigned memory locations with a valid data to make sure they have values).

Program:

```
MOV AX, 5000H  ; Store the segment address 5000H in AX.
MOV DS, AX     ; Initialize DS with the segment address 5000H.
MOV ES, AX     ; Initialize ES with the segment address 5000H.
MOV SI, 1000H  ; Initialize SI with the offset of the source's starting
                 address.
MOV DI, 3000H  ; Initialize DI with the offset of the destination
                 address.
MOV CX, 100    ; Initialize CX with the number of words in the string
                 (decimal value of 100 or 64H).
CLD            ; Clear the D flag for auto-increment mode.
REP MOVSW      ; Execute MOVSW instruction CX times.
HLT            ; Terminate program execution.
```

Note: In this program, the segment addresses of the source and destination are the same, and hence DS and ES registers are loaded with the same value. If they are different, ES and DS registers are loaded with the segment address of the destination and source, respectively. As D is 0, every time MOVSW is executed, the SI and DI registers are incremented by 2 to point to the next word in the string.

Example 2 of 8086 Assembly Programming Using MASM

- Write an ALP to find factorial of number for 8086.

```
        MOV AX, 05H
        MOV CX, AX
        DEC CX
Back:   MUL CX
        LOOP back          ; results stored in AX
        MOV [D000], AX     ; to store the result at D000H
        HLT
```

Example 3 of 8086 Assembly Programming Using MASM

- The 8 data bytes are stored from memory location E000H to E007H. Write 8086 ALP to transfer the block of data to new location B001H to B008H.

```
        MOV CX, 0008H
        MOV BX, E000H
        MOV AX, B001H
Back:   MOV DL, [BX]
        MOV [AX], DL
        Loop Back
        HLT
```

Main Sources for these slides

- *K. R. Irvine. Assembly Language for x86 Processors, 8th edition, Prentice-Hall (Pearson Education), June 2019. ISBN: 978-0135381656.*

- *B. Dang, A. Gazet, E. Bachaalany. Practical Reverse Engineering: x86, x64, ARM, Windows® Kernel, Reversing Tools, and Obfuscation. John Wiley & Sons, June 2014. ISBN: 978-1-118-78731-1*

- *Qasem Abu Al-Haija, "Microprocessor Systems", King Faisal University, Saudi Arabia*

- *Ghassan Issa, "Computer Organization", Petra University, Jordan.*

Thank you