Mansoura University
Faculty of Computers and Information
Department of Computer Science
First Semester: 2020-2021

[CS214P] Assembly Language: Chapter 5

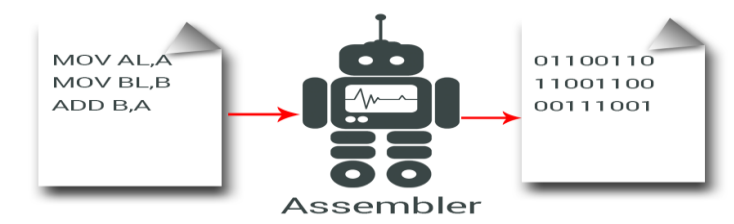
Grade: Third Year (Computer Science)

Sara El-Metwally, Ph.D.

Faculty of Computers and Information,

Mansoura University,

Egypt.



Computer Science Department
Faculty of Computers and Information
Mansoura University

Assembly Language

"Symbolic Instructions and Addressing"

Sara El-Metwally, Ph.D.
Faculty of Computers and Information,
Mansoura University, Egypt.

Email: sarah_almetwally4@mans.edu.eg
sara.elmetwally.2007@gmail.com

Data Transfer Instructions

- **The MOV Instruction**
  - **Transfers data from the address of the second operand to the address of the first operand.**
  - **The operands must agree in size.**

[label:]	MOV	R/M, R/M/I

```
; Registers Moves

BYTED DB ?
WORDD DW ?

MOV EDX, ECX    ;R--->R
MOV ES,AX       ;R---->SR
MOV BYTED,DH    ;R--->M Direct
MOV [DI],BX     ;R--->M Indirect
```

Data Transfer Instructions

```
; Immediate Moves

BYTED DB ?
WORDD DW ?

MOV CX,40H    ;I--->R
MOV BYTED,25  ;I--->M Direct
MOV WORDD[BX],16H ;I--->M Indirect
```

```
; Direct Memory Moves

BYTED DB ?
WORDD DW ?

MOV CH,BYTED ;M---->R Direct
MOV CX,WORDD[BX];M---->RIndirect
```

```
; Segment Register Moves

BYTED DB ?
WORDD DW ?

MOV AX,DS  ;SR----> R
MOV WORDD,DS;SR---->M
```

Data Transfer Instructions

```
MOV DL,WORD_VAL
MOV CX, BYTE_VAL
MOV WORD_VAL,EBX
MOV BYTE_VAL2,BYTE_VAL1
MOV ES,225
MOV ES,DS
```

Move-and-Fill instructions

- **MOVSX and MOVZX**
  - **Fix the problem of the destination must be the same length as the source.**
  - **(80386+) facilitate transferring data from a byte or word source to a word or double word destination.**

[label:]	MOVSX MOVZX	R/M, R/M/I

```
.386

MOVSX CX, 10110000B   ; CX = 11111111 10110000
MOVZX CX, 10110000B   ; CX = 00000000 10110000
```

Data Transfer Instructions

- **The XCHG Instruction**
  - **Swap data items**

[label:]	XCHG	R/M, R/M

```
wordd1 DW 1512
wordd2 DW 2030
XCHG CL,BH
XCHG CX,wordd1
XCHG wordd2,wordd1
```

Data Transfer Instructions

- The LEA Instruction
  - Initializing a register with an offset address.
  - BX, DI, and SI.

[label:]	LEA	R,M

```
DATA DB 25 DUP (?)
DBYTE DB ?
LEA BX, DATA  ; load offset address
MOV DBYTE,[BX]

MOV BX, OFFSET DATA ; load offset address
```

Arithmetic Instructions

- The INC/DEC Instruction
  - Increment/decrement the contents of registers and memory location by 1.
  - Requires only one operand.
  - OF, SF, and ZF are affected by INC/DEC.
  - Conditional jump instructions may test these conditions.
  - INC FFH? DEC 00H?

[label:]	INC/DEC	R/M

Arithmetic Instructions

- The ADD/SUB Instruction
  - AF,CF,PF,OF, SF, and ZF are affected.

[label:]	ADD/SUB	R/M, R/M/I

```
ADD AX,CX
ADD EBX, WORDD
SUB BL,10
```

Repetitive MOV operations

Sara2.asm*

```
page 60,132
title Hellow World from assembly
;-----------------------------
.MODEL SMALL
.STACK 64
.DATA
 DATA1 DB 'Hellow World'
 DATA2 DB 12 DUP('*'),'$'
.CODE
   MAIN PROC FAR
        MOV AX,@data
        MOV DS, AX

        MOV CX,12
        LEA SI,DATA1
        LEA DI,DATA2
   A20:
        MOV AL,[SI]
        MOV [DI],AL
        INC SI
        INC DI
        DEC CX
        JNZ A20

        MOV AH,09H
        LEA DX,DATA2
        INT 21H

        MOV AX,4C00H
        INT 21H

   MAIN ENDP
        END MAIN
```

GUI Turbo Assembler x64

```
Hellow World


Program successfully executed !
Press any key to continue.
_
```

8086 Instruction Set

www.electronics.dit.ie/staff/tscarff/8086_instruction_set/8086_instruction_set.html

Complete 8086 instruction set

Quick reference:

	CMPSB				MOV
AAA	CMPSW	JAE	JNBE	JPO	MOVSB
AAD	CWD	JB	JNC	JS	MOVSW
AAM	DAA	JBE	JNE	JZ	MUL
AAS	DAS	JC	JNG	LAHF	NEG
ADC	DEC	JCXZ	JNGE	LDS	NOP
ADD	DIV	JE	JNL	LEA	NOT
AND	HLT	JG	JNLE	LES	OR
CALL	IDIV	JGE	JNO	LODSB	OUT
CBW	IMUL	JL	JNP	LODSW	POP
CLC	IN	JLE	JNS	LOOP	POPA
CLD	INC	JMP	JNZ	LOOPE	POPF
CLI	INT	JNA	JO	LOOPNE	PUSH
CMC	INTO	JNAE	JP	LOOPNZ	PUSHA
CMP	IRET	JNB	JPE	LOOPZ	PUSHF
	JA				RCL

8086 Instruction Set (DIV)

DIV	REG memory

Unsigned divide.

Algorithm:

when operand is a **byte**:
AL = AX / operand
AH = remainder (modulus)

when operand is a **word**:
AX = (DX AX) / operand
DX = remainder (modulus)

Example:

```
MOV AX, 203   ; AX = 00CBh
MOV BL, 4
DIV BL        ; AL = 50 (32h), AH = 3
RET
```

C	Z	S	O	P	A

ODD/EVEN Program

```
page 60,132
title ODD_EVEN_PROGRAM
;-----------------------------
.MODEL SMALL
.STACK 64
.DATA
 MS1 DB 'Enter the number', '$'
 MS2 DB 'Number is odd','$'
 MS3 DB 'Number is even','$'
 nl  DB 0dh,0ah, '$'
.CODE
   MAIN PROC FAR
        MOV AX,@data
        MOV DS, AX
        MOV ES, AX

        LEA SI, MS1
        CALL PRINT
        CALL NEW_LINE
```

ODD/EVEN Program

```
21
22          MOV AH,01H
23          INT 21H
24          MOV BL,2
25          DIV BL
26          CMP AH,0
27          JNE ODD
28
29          CALL NEW_LINE
30
31          LEA SI, MS3
32          CALL PRINT
33          JMP Bye
34
35        ODD:
36          CALL NEW_LINE
37          LEA SI, MS2
38          CALL PRINT
39
40        Bye:
41          MOV AX,4C00H
42          INT 21H
43
44    MAIN ENDP
```

ODD/EVEN Program

```
PRINT PROC NEAR
      MOV AH,09H
      MOV DX,SI
      INT 21H
      RET
PRINT ENDP

NEW_LINE PROC NEAR
   LEA DX,nl
   MOV AH,09H
   INT 21H
   RET
NEW_LINE ENDP

      END MAIN
```

GUI Turbo Assembler x64

```
Enter the number
6
Number is even


Program successfully executed !
Press any key to continue.
```

INT Instruction

- The INT Instruction
  - INT enables a program interrupt .
  - Pushes the contents of the flag register onto the stack.
  - Clears the interrupt and trap flags.
  - Pushes CS register onto the stack.
  - Pushes IP onto the stack.
  - Perform the required operation.
  - To return, the operation issues an IRET, which pops the registers off the stack.
  - The restored CS:IP causes a return to the instruction immediately following the INT.

Addressing modes
(Register Addressing)

- It is fastest type of operation because there is no reference to memory (R,R).

```
1
2  MOV DX, WORD_MEM
3  MOV WORD_MEM, CX
4  MOV AX, DX
5
6
```

Addressing modes
(Immediate Addressing)

- It is a constant value should be the second operand.
- The destination field (first operand) defines the length of the data.

```
Byte_VAL     DB 150
WORD_VAL     DW 300

SUB Byte_VAL, 50
MOV WORD_VAL,40H
MOV AX,0245H
```

Addressing modes
(Immediate Addressing)

- It is a constant value should be the second operand.
- The destination field (first operand) defines the length of the data.

```
Byte_VAL     DB 150
WORD_VAL     DW 300

SUB Byte_VAL, 50
MOV WORD_VAL,40H
MOV AX,0245H
```

```
MOV AL,0245H
ADD AX,48H
```

Addressing modes
(Direct Memory Addressing)

- One of operands is R and the other is M.
- MOVS and CMPS only two instructions that allow both operands to address memory directly.

```
SUB Byte_VAL, DL
MOV BX,WORD_VAL
```

Addressing modes
(Direct -Offset addressing)

```
1
2  Byte_TABEL DB 12,15,16,22
3  WORD_TABEL DW 163,227,485
4
5  MOV CL,Byte_TABEL[2]
6  MOV CL,Byte_TABEL+2
7
8
9  MOV CX,WORD_TABEL[4]
10 MOV CX,WORD_TABEL+4
11
12
```

Addressing modes
(Indirect Memory Addressing)

- BX, BP,DI,SI.
- DS:BX, DS:SI, DS:DI, and SS:BP.
- When the first operand contains an indirect address, the second operand should be a register or an immediate value.

```
Byte_Val DB 50
LEA BX,Byte_Val
MOV [BX],CL

ADD CL,[BX]
MOV BYTE PTR [DI],25
ADD [BP],CL


MOV CX,DS:[38B0H]
```

Addressing modes
(Base Displacement Addressing)

- (BX, BP,DI,SI) + displacement (a number or offset value) to form an effective address.

```
Byte_TBL DB 365 DUP(?)
LEA BX,Byte_TBL
MOV BYTE PTR [BX+2],0    ; MOV 0 to Byte_TBL+2

ADD CL,[DI+12]
SUB Byte_TBL[SI],25
MOV Byte_TBL[DI],DL
```

Addressing modes
(Base-Index Addressing)

- (BX, BP,DI,SI) with each other.

```
0
1 MOV AX,[BX+SI]
2 ADD [BX+DI],CL
3
4
```

Addressing modes
(Base-Index with Displacement Addressing)

```
MOV AX,[BX+SI+10] ;or 10[BX+SI]
MOV CL,DATA_TBL[BX+DI]; or [ BX+DI+DATA_TBL]
```

Segment override prefix

```
MOV DX,ES:[BX]
MOV ES:[SI+36],CL
```

- The assembler generates object code with the override operator inserted as a 1-byte prefix (26H) immediately preceding the instruction.

```
ES: MOV DX, [BX]
ES: MOV [SI+36],CL
```

NEAR and FAR addresses

- NEAR Address: only 16-bit offset portion of an address.
- An instruction that references a near address assumes the current segments are DS (Data) and CS (Instruction).
- FAR Address: segment :offset.
- An instruction can reference the far address from the current segment or in another segment.

Enter & Display a Number with Two Digits

Program

```
;-----------------------------
.MODEL SMALL
.STACK 64
.DATA
 MS1 DB 'Enter any Number with Two Digits', '$'
 MS2 DB 'You have Entered','$'
 N1  DB 0
 N2  DB 0
 Ten DB 10
 T1  DB 0
 nl  DB 0dh,0ah, '$'
.CODE
   MAIN PROC FAR
        MOV AX,@data
        MOV DS, AX
        MOV ES, AX
        LEA SI, MS1
        CALL PRINT
        CALL NEW_LINE

        CALL READ
        SUB AL,48
        MOV N1,AL

        CALL READ
        SUB AL,48
        MOV N2,AL
```

48	30	00110000	0
49	31	00110001	1
50	32	00110010	2
51	33	00110011	3
52	34	00110100	4
53	35	00110101	5
54	36	00110110	6
55	37	00110111	7
56	38	00111000	8
57	39	00111001	9

Enter & Display a Number with Two Digits Program

```
        MOV AL,N1
        MUL Ten
        ADD AL,N2

        CALL NEW_LINE

        LEA SI,MS2
        CALL PRINT

        CALL NEW_LINE

        MOV AH,00
        DIV Ten
        MOV T1,AH ; remainder
        MOV DL,AL
        CALL PRINT_C
        MOV DL,T1
        CALL PRINT_C

       Bye:
        MOV AX,4C00H
        INT 21H


   MAIN ENDP
```

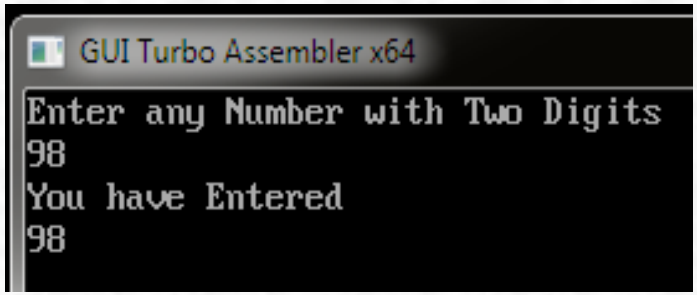
Enter & Display a Number with Two Digits Program

```
PRINT_C PROC NEAR
      MOV AH,02H
      ADD DL,48
      INT 21H
      RET
 PRINT_C ENDP

 PRINT PROC NEAR
      MOV AH,09H
      MOV DX,SI
      INT 21H
      RET
 PRINT ENDP

 NEW_LINE PROC NEAR
    LEA DX,nl
    MOV AH,09H
    INT 21H
    RET
 NEW_LINE ENDP

 READ PROC NEAR
    MOV AH,01h
    INT 21h
    RET
 READ ENDP
```



Print Numbers from 1 to 100 Program

```
.MODEL SMALL
.STACK 100
.DATA
 NUM DW ?
 N   DB 4 DUP('*'),'$'
 nl  DB 0dh,0ah, '$'
.CODE
   MAIN PROC FAR
        MOV AX,@data
        MOV DS, AX
        MOV ES, AX
        MOV NUM, 0
      Do:
        CMP NUM,100
        JBE Go
        JMP Bye

      Go:
        MOV SI,offset N ;What is another Way?
        MOV AX,NUM
        CALL Get_Num
        CALL Print_Num
        CALL NEW_LINE
        INC NUM
        JMP DO
      Bye:
         MOV AX,4C00H
         INT 21H

   MAIN ENDP
```

Print Numbers from 1 to 100 Program

```
Get_Num PROC NEAR
        MOV BX,10
        MOV CX,0
        extract_one_digit:
        MOV DX,0
        DIV BX
        PUSH DX
        INC CX
        CMP AX,0
        JNE extract_one_digit

        adjust_one_digit:
        POP DX
        ADD DL,48
        MOV [SI], DL
        INC SI
        LOOP adjust_one_digit
        RET
Get_Num ENDP
```

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9

Print Numbers from 1 to 100 Program

```
NEW_LINE PROC NEAR
   LEA DX,nl
   MOV AH,09H
   INT 21H
   RET
NEW_LINE ENDP
Print_Num PROC NEAR
   LEA DX,N
   MOV AH,09H
   INT 21H
   RET
Print_Num ENDP
```