CSE-3108

Spring, 2016

Connecting MDA-8086 to PC and Basic Assembly Programming

What did we learn on the last week?

- Introduction to 8086 and MDA-8086
- Getting familiar with the MDA-8086 keyboard
- Introduction to segmented memory, segment and offset registers
- *[Segment, Offset]* → *Physical address* mapping
- Inserting data into desired address of the MDA-8086 RAM(or physical memory)
- Writing machine codes for simple assembly programs
- Executing the written assembly program step by step using the on-board `STP` key

Objective of today's lab

Today you will learn to connect the MDA-8086 with your PC through a serial port. You have already learned assembly programming for Intel-8086 on a previous course. Today you need to-

- Write assembly program(.ASM extension) on your PC
- From the assembly program, generate the machine code, which is the hex file(.ABS extension)
- Upload the hex file to MDA-8086's RAM
- Execute the program and test the output of the program from the PC

We shall use MASM(Microsoft Assembler) as the assembler program

A simple program to add 2 numbers

```
CODE SEGMENT            ; We start a segment named 'CODE'
     ASSUME CS:CODE     ; We tell the assembler to point the CS register to the segment named 'CODE'
     ORG 1000H          ; ORG is an indication on where to put the next piece of code/data, related
                        ; to the current segment

     MOV AX, 1234H
     MOV BX, 4321H
     ADD AX, BX
     INT 3

CODE ENDS
END
```

How to Connect PC with MDA-8086

2 Set up MASM ASSEMBLER like follows

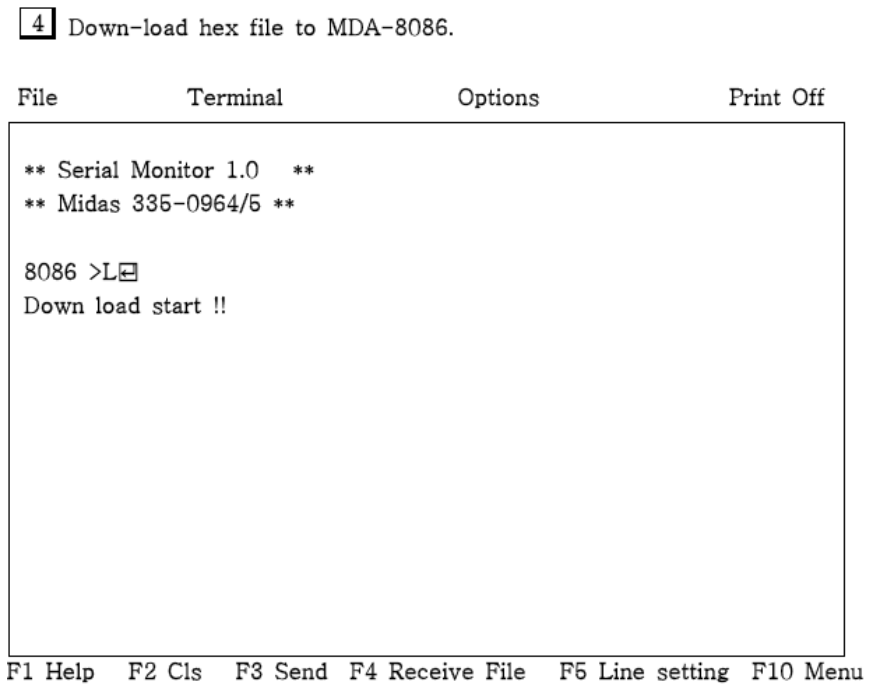
```
C:₩8086>MASM↵
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Misrosoft Corp 1981, 1988. All right reserved.

     Source filename [.ASM]:EX2↵
     Object filename [C:EX2.OBJ]:↵
     Source listing  [NUL.LST]:EX2↵
     Cross reference [NUL.CRF]:↵

47838 + 452253 Bytes symbol space free
  0 Warning Errors
  0 Severe  Errors

C:₩8086>
```

How to Connect PC with MDA-8086

3 Make HEX(ABS) file.

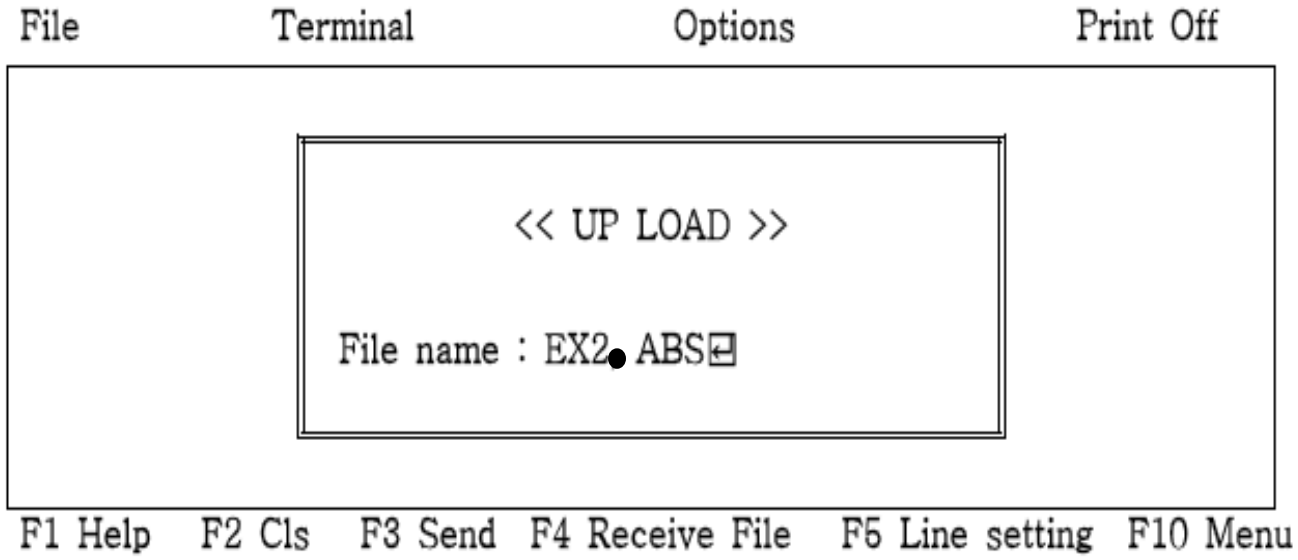
```
C:₩8086>LOD186↵
Paragon LOD186 Loader - Version 4.0h
Copyright (C) 1983 - 1986 Microtec Research Inc.
ALL RIGHT RESERVED.

Object/Command File                [.OBJ]:EX2↵
Output Object File              [C:EX2.ABS]:↵
Map Filename                     [C:NUL.MAP]:↵

**LOAD COMPLETE

C:₩8086>
```

How to Connect PC with MDA-8086



How to Connect PC with MDA-8086

1 Strike PgUp or F3 key in computer, and then like following will be displayed.



Some useful commands

- The **L** command moves object data in hex format from an external devices to memory. We should execute this command first
- **G** to run the loaded program. Entering this command asks for the segment and offset address to the beginning of the code
- **T** for executing the next instruction. This is similar to the '*STP*' key.

Some useful commands(cont.)

- **E** for RAM content modification

```
                Segment   Offset
                   ↓        ↓
8086 >E 0000:1000↵
0000:1000 FF ? 11↵
0000:1001 FF ? 22↵
0000:1002 FF ? 33↵
0000:1003 FF ? 44↵
0000:1004 FF ? 55↵
0000:1005 FF ? /  ↵  ← (Offset decrement)
0000:1004 55 ? /  ↵
0000:1003 44 ? .  ↵   ← (Escaping command)
```

Some useful commands(cont.)

- **D** for displaying RAM content

```
          Segment   Offset
             ↓        ↓
8086 >D 0000:1000↵
0000:1000 11 22 33 44 55 FF FF FF - FF FF FF FF FF FF FF FF    ."3DU.....
0000:1010 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
0000:1020 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
0000:1030 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
0000:1040 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
0000:1050 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
0000:1060 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
0000:1070 FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF    ........
8086 >                                        Display the ASCII code to data
```

Some useful commands(cont.)

- **R** for displaying register contents

```
8086 >R↵
        AX=0000  BX=0000  CX=0000  DX=0000
        SP=0540  BP=0000  SI=0000  DI=0000
        DS=0000  ES=0000  SS=0000  CS=0000
        IP=1000  FL=0000  = . . . . . .
```

Add 2 numbers and save the result on RAM1

```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 1000H

    MOV AX, 1000H
    MOV DS, AX
    MOV SI, 0H                  ; DS:SI = 1000H:0000H
    MOV AX, 1234H
    MOV BX, 4321H
    ADD AX, BX
    MOV DS:[SI], AL             ; Insert 'AL' into 1000H:0000H
    INC SI                      ; SI = 1000H
    MOV DS:[SI], AH             ; Insert 'AX' into 1000H:0001H
    INT 3

CODE ENDS
END
```

Add 2 numbers and save the result on RAM_2

```
CODE SEGMENT
     ASSUME CS:CODE
     ORG 1000H

     MOV AX, 1000H
     MOV DS, AX
     MOV SI, 0H                    ; DS:SI = 1000H:0000H
     MOV AX, 1234H
     MOV BX, 4321H
     ADD AX, BX
     MOV WORD PTR DS:[SI], AX      ; Here, DS:[SI] works as a 2-byte pointer, or word pointer
     INT 3

CODE ENDS
END
```

If-Else in assembly

- We cannot write if-else control flow that easily in assembly
- We have to use some instructions to set the flags(for example `CMP`, `TEST` instructions), and then use jump instructions (`JMP`, `JE`, `JNE`, `JG`, `JGE`, `JL`, `JLE`, `JZ`, `JNZ` etc) to create if-else logic.
- Learn bitwise and shift instructions clearly. These would help you on later lab-works as well
- A simple parity checker(even-odd checker) is demonstrated on the next slide

Parity checker

```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 1000H

    MOV AX, 1000H
    MOV DS, AX
    MOV SI, 0H              ; DS:SI = 1000H:0000H
    MOV AX, 005AH           ; Load the number of which we want to check the parity of, into AX
    TEST AX, 0001H          ; TEST instruction is similar to AND instruction
    JZ EVEN
    JNZ ODD
EVEN:
    MOV BL, 0H
    JMP DONE
ODD:
    MOV BL, 1H
    JMP DONE
DONE:
    MOV DS:[SI], BL         ; Write the result in memory location [1000:0000]
    INT 3

CODE ENDS
END
```

Bitwise operation instructions

Instruction	Format
AND	AND operand1, operand2
OR	OR operand1, operand2
XOR	XOR operand1, operand2
TEST	TEST operand1, operand2
NOT	NOT operand1

- The first operand in all the cases could be either in register or in memory
- The second operand could be either in register/memory or an immediate (constant) value
- However, memory-to-memory operations are not possible
- These instructions compare or match individual bits of the operands and set the CF, OF, PF, SF and ZF flags

Shift and rotate instructions

Instruction	Syntax	Note
SHL/SAL	`SHL destination, shift_amount`	`shift_amount` can be a constant / `CL`.
SHR	`SHR destination, shift_amount`	Shift write operation. MSB become 0 after.
SAR	`SAR destination, shift_amount`	Shift write operation. MSB retains original data.
ROL	`ROL destination, rotate_amount`	`rotate_amount` can be a constant / `CL`.
ROR	`ROR destination, rotate_amount`	Similar to ROR
RCL	`RCL destination, rotate_amount`	Considers `CF`(Carry Flag) as an extension of "`destination`"
RCR	`RCR destination, rotate_amount`	same

Figure 7.2 SHL and SAL

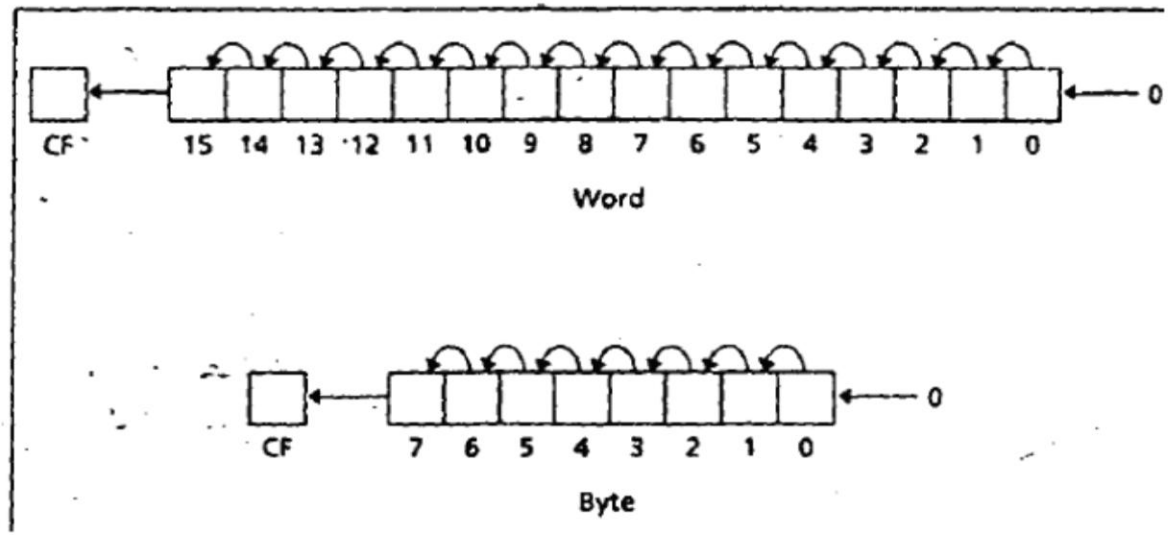


Figure 7.3 SHR

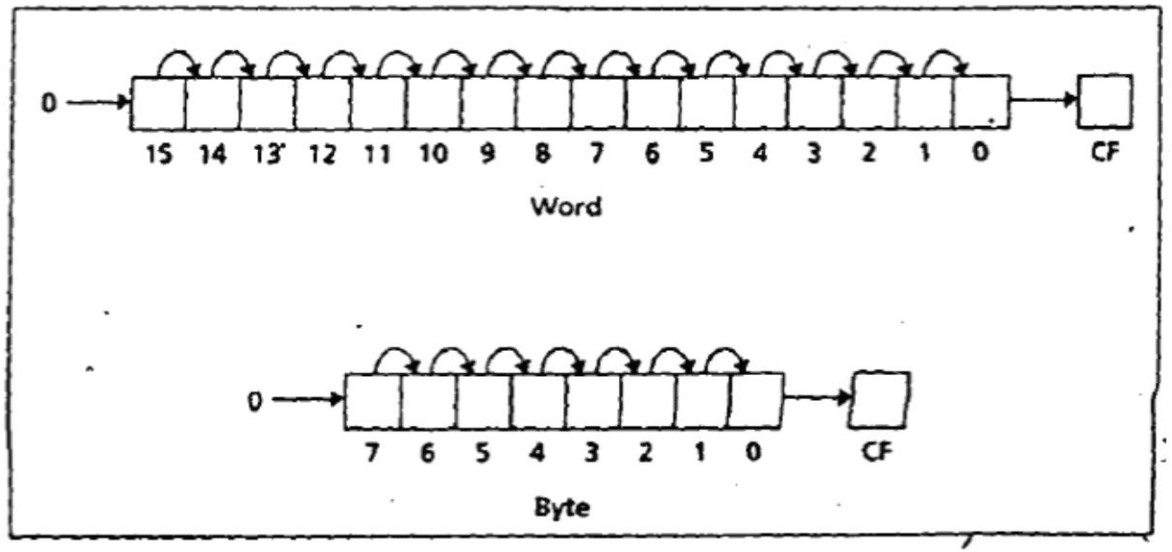
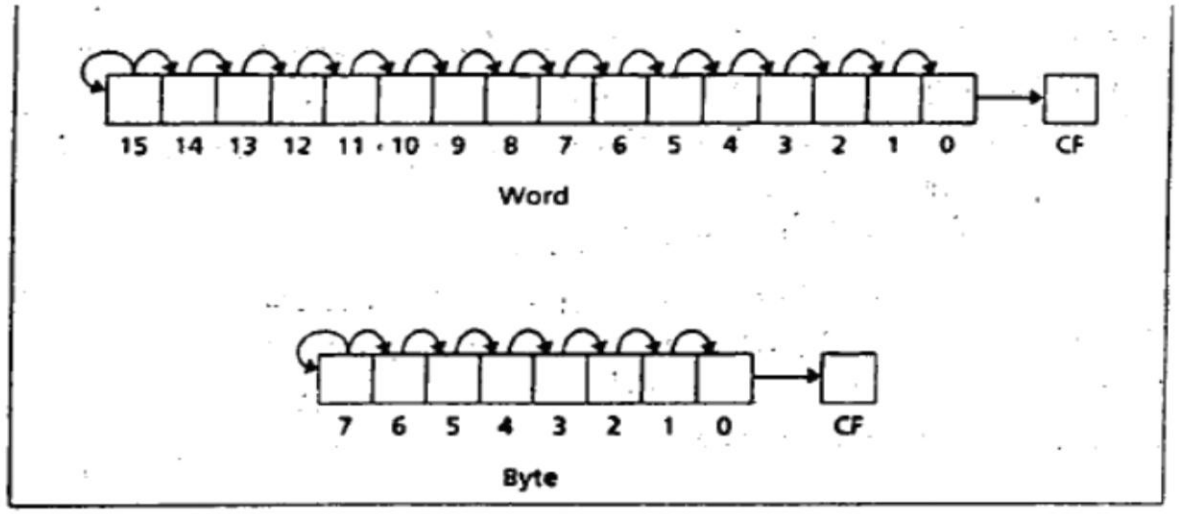
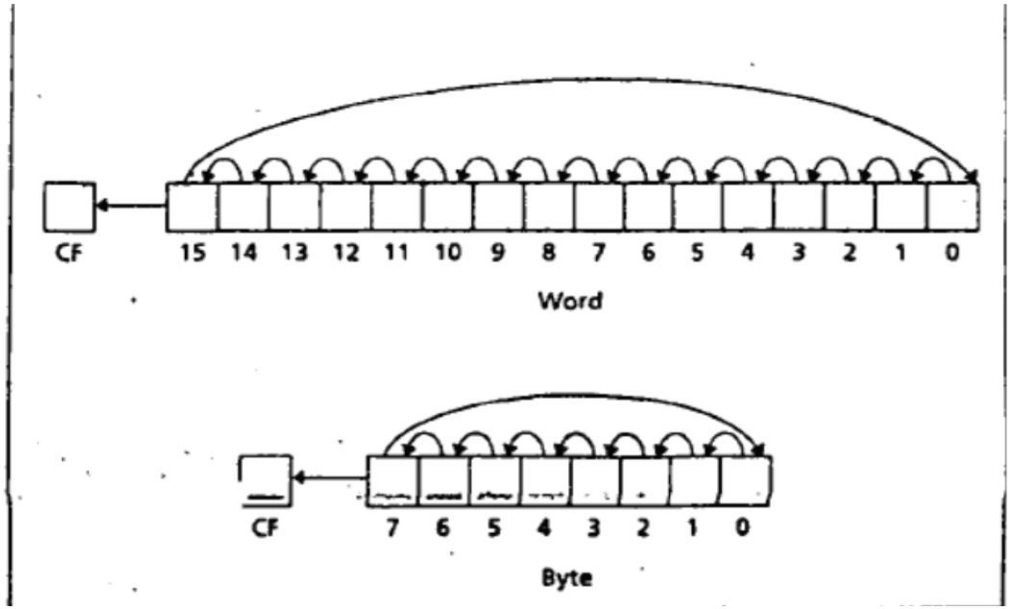


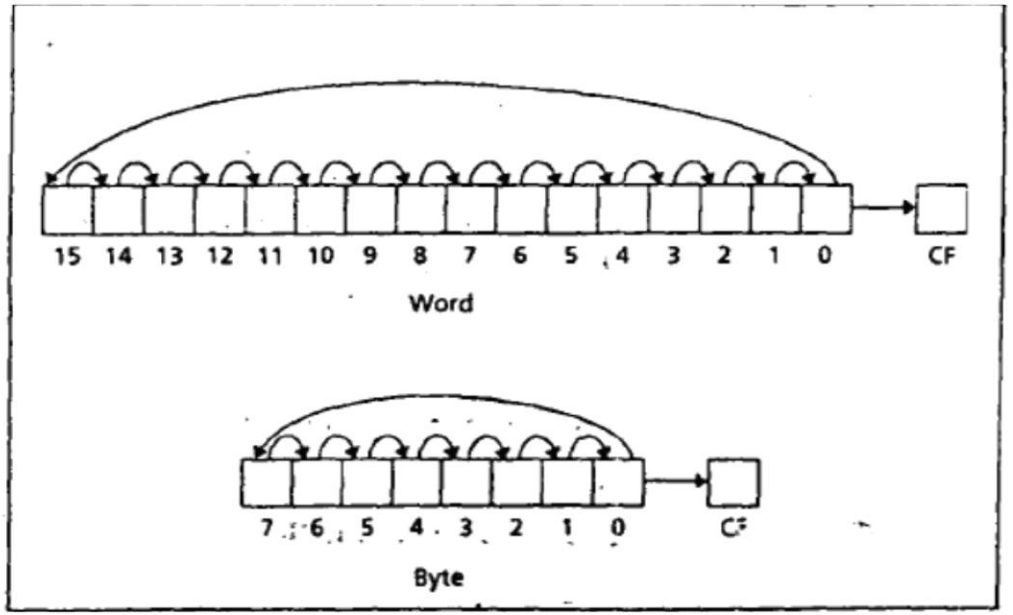
Figure 7.4 SAR



igure 7.5 ROL



igure 7.6 ROR



Loops in assembly

- Loops can also be implemented by combining the use of some instructions to set the flags(for example *CMP*, *TEST* instructions), and jump instructions (*JMP*, *JE*, *JNE*, *JG*, *JGE*, *JL*, *JLE*, *JZ*, *JNZ* etc).
- There is also an instruction called "*LOOP*" that works with the register CX.
- Let's assume we want to write an assembly program to compute the sum of the series *1+2+3+ … + n*
- The following two slides demonstrate two solutions to this problem using looping

Compute *1+2+3+ … + n* solution1

```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 1000H

    MOV AX, 1000H
    MOV DS, AX
    MOV SI, 0H    ; DS:SI = 1000H:0000H
    MOV CX, 9H    ; Load the value of 'n' in CX
    MOV AX, 0H    ; Initially SUM = 0
LOOPLABEL:
    ADD AX, CX
    DEC CX
    CMP CX, 0H
    JNE LOOPLABEL

    MOV WORD PTR DS:[SI], AX
    INT 3

CODE ENDS
END
```

Compute 1+2+3+ … + n solution2

```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 1000H

    MOV AX, 1000H
    MOV DS, AX
    MOV SI, 0H          ; DS:SI = 1000H:0000H
    MOV CX, 9H          ; Load the value of 'n' in CX
    MOV AX, 0H          ; Initially SUM = 0
LOOPLABEL:
    ADD AX, CX
    LOOP LOOPLABEL      ; LOOP instruction decrements CX, checks if the decremented value is 0,
                        ; and if it is 0, then jumps to the associated label

    MOV WORD PTR DS:[SI], AX
    INT 3

CODE ENDS
END
```

Procedures or functions

- Just like in case of the high level programming languages, procedures or subroutines are just as important in assembly language
- Procedure calls utilize the stack memory segment
- We can pass arguments of a procedures by pushing them onto stack
- We can also return values from parameters using stack memory
- Whenever a procedure is **CALL**ed, the return address(which is the instruction next to the procedure call instruction) is pushed onto stack. When the program **RET**urns from that procedure, the top of the stack is popped from the stack to the **IP** register.

Write a procedure for the sum of the series *1+2+3+ … + n*


```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 1000H

    MOV AX, 1000H
    MOV DS, AX
    MOV SI, 0H          ; DS:SI = 1000H:0000H
    MOV CX, 09H         ; Load the value of 'n1' to CX
    CALL series_sum     ; Now AX hold the value of the
                        ; sum 1+2+...n1
    MOV WORD PTR DS:[SI], AX
    MOV CX, 64H         ; Load the value of 'n2' to CX
    CALL series_sum     ; Now AX hold the value of the
                        ; sum 1+2+...n2
    ADD SI, 2H
    MOV WORD PTR DS:[SI], AX
    INT 3
```

```
series_sum:
    PUSH CX ; Make a backup of the
            ; value stored in CX during
            ; procedure call
    MOV AX, 0H
LOOP_LABEL:
    ADD AX, CX
    LOOP LOOP_LABEL
    POP CX
    RET

CODE ENDS
END
```

[Recursion] Compute factorial

```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 1000H

    MOV AX, 1000H
    MOV DS, AX
    MOV SI, 0H
    MOV CX, 6H ;Calculate factorial(CX)
    CALL fact
    MOV WORD PTR DS:[SI], AX
    INT 3
```

```
fact:
    CMP CX, 0H
    JNE CALCULATE
    MOV AX, 1H
    JMP RETURN_LABEL
CALCULATE:
    DEC CX
    CALL fact
    INC CX
    MUL CX
RETURN_LABEL:
    RET


CODE ENDS
END
```