Chapter 6
Assembly (1)

Objectives

After studying this part, you should be able to:

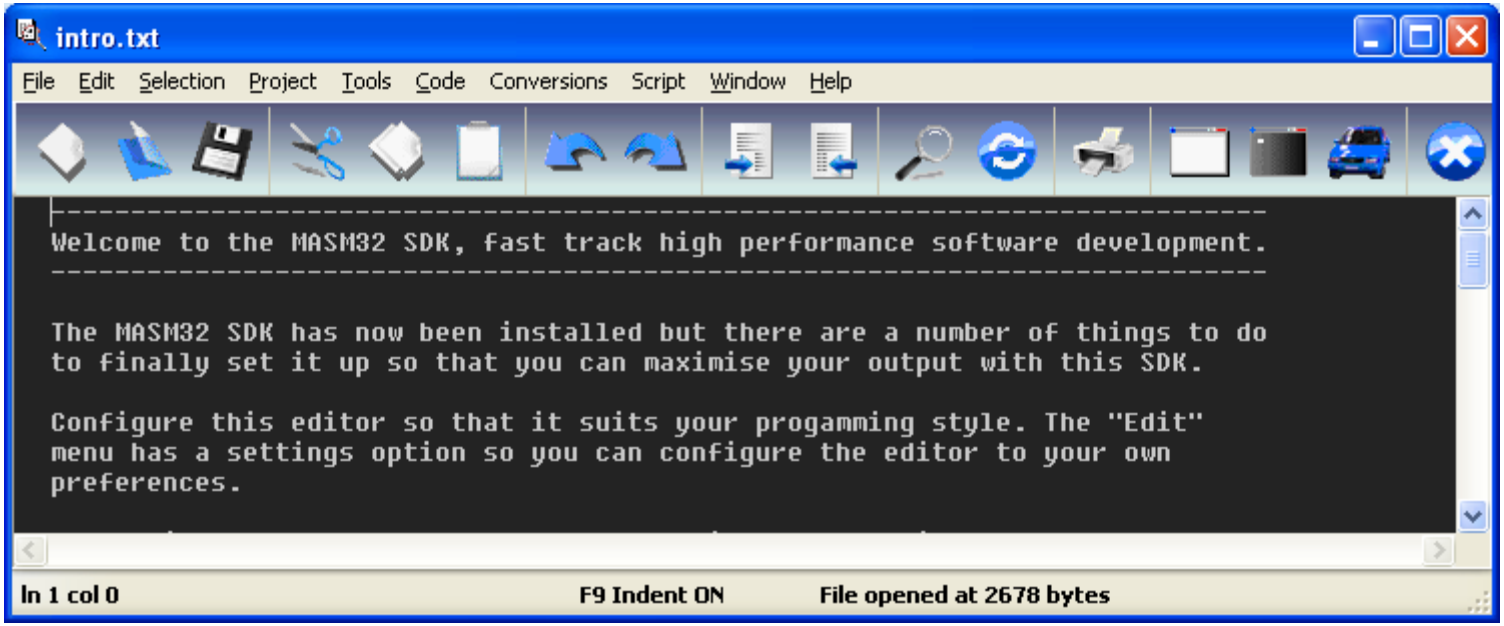
- Familiarize yourself with the assembly language, a low level language
- Understand how a program is compiled
- Develop some basic console applications

Contents

- 1- Install 32/64-bit MASM – MS Macro Assembly
- 2- MASM Integrated Development Environment(IDE)
- 3- Introduction to Microsoft Macro Assembly Language
- Some sample programs

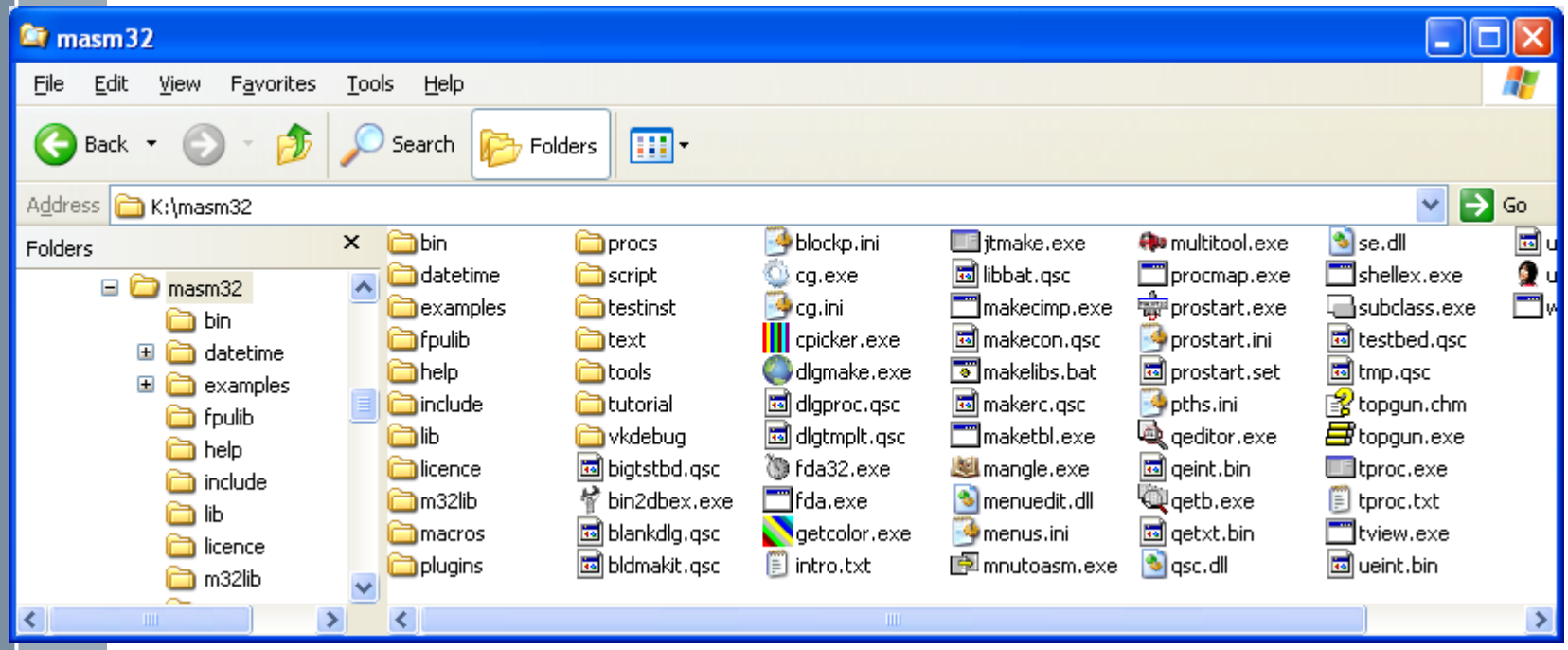
1- Install 32-bit MSAM

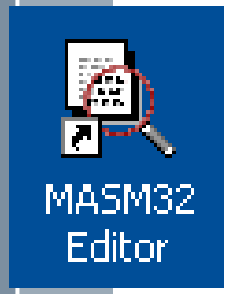
- Microsoft Macro Assembler: an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows. Beginning with MASM 8.0 there are two versions of the assembler - one for 16-bit and 32-bit assembly sources, and another (**ML64**) for 64-bit sources only (Wiki)
- Unzip: masm32v11r.zip → Install.exe → Run for installation
- Interface after installation:



Install 32-bit MSAM…

› Installed Contents:

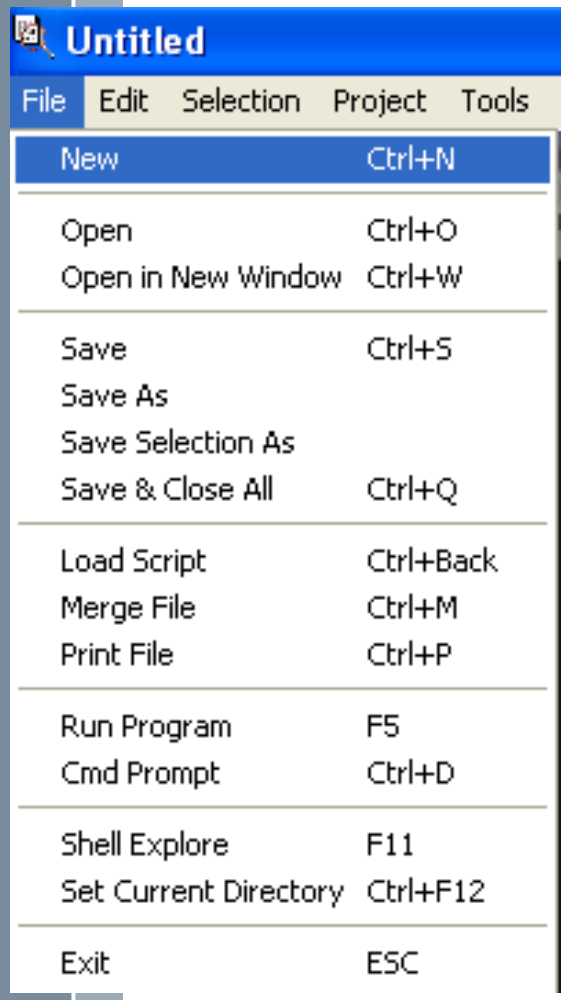




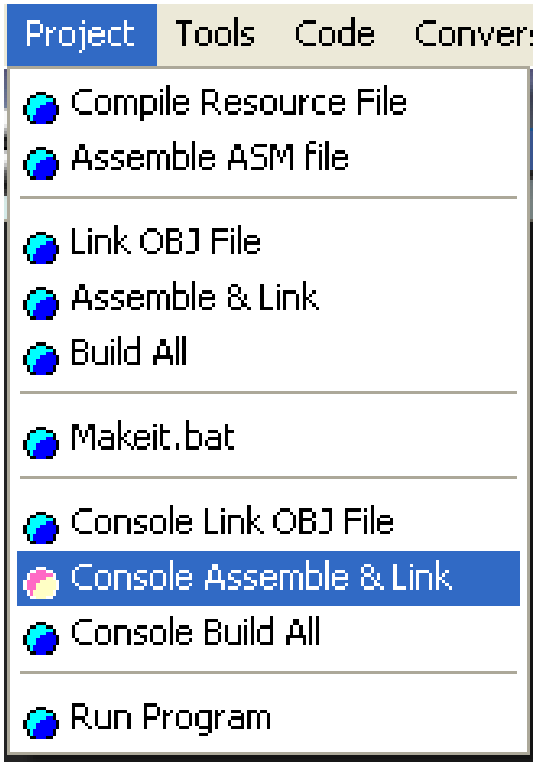
Desktop Icon of MASM, executable file: **qeditor.exe**
Compiler: bin/ml.exe (32 bit), ml64.exe (64 bit)

You should create a folder as a storage of your exercises

2- MASM Integrated Development Environment

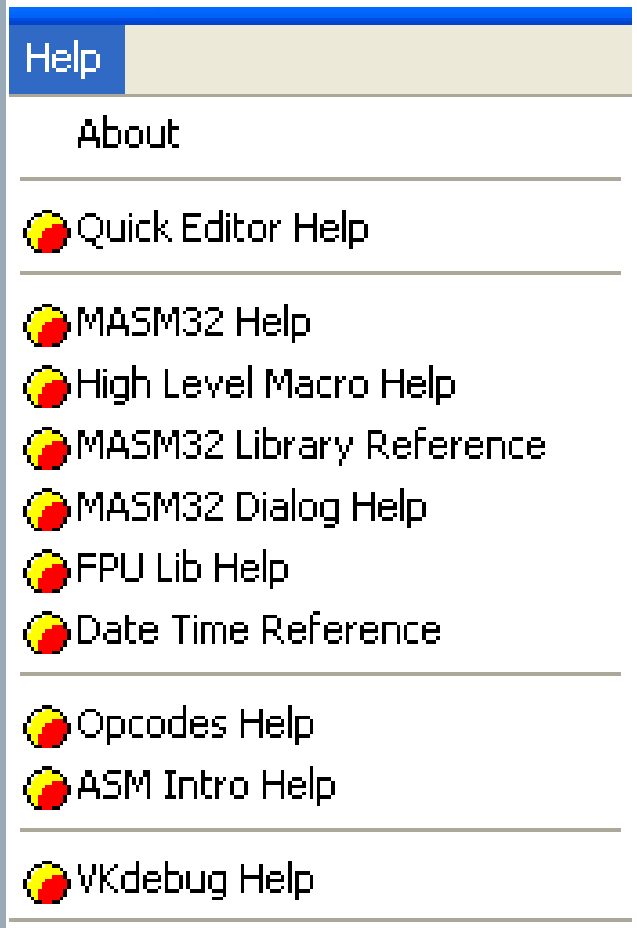


Menu **file** allows user working with files, run program,…



Menu **Project** allows user compiling program,…

2- MASM Integrated Development Environment



Menu **Help** allows user referencing to relative topics:

Using editor

Build-in Libraries in MASM

Opcodes of Intel CPU
Syntaxes of MASM language

3- Introduction to MS Assembly

Demo 1: Write an Assembly program that displays the string 'Hello World' on the screen:

The **flat memory model** is a *non-segmented* configuration available in 32-bit operating systems.

```
; «« Comment begins with ';' to the end of a line
; From masm32\tutorial\console\demo1
;
; Build this with the "Project" menu using
; "Console Assemble and Link"
; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

    .486                                      ; create 32 bit code
    .model flat, stdcall                      ; 32 bit memory model
    option casemap :none                      ; case sensitive

    include \masm32\include\windows.inc       ; always first
    include \masm32\macros\macros.asm         ; MASM support macros

; -----------------------------------------------------------------
; include files that have MASM format prototypes for function calls
; -----------------------------------------------------------------
    include \masm32\include\masm32.inc
    include \masm32\include\gdi32.inc
    include \masm32\include\user32.inc
    include \masm32\include\kernel32.inc

; ------------------------------------------------
; Library files that have definitions for function exports
; and tested reliable prebuilt code.
; ------------------------------------------------
    includelib \masm32\lib\masm32.lib
    includelib \masm32\lib\gdi32.lib
    includelib \masm32\lib\user32.lib
    includelib \masm32\lib\kernel32.lib

    .code                          ; Tell MASM where the code starts

    start:                             ; The CODE entry point to the program
        print chr$("Hello world!",13,10) ; 13: carriage return, 10: new line
    exit                               ; exit the program

; -------------------------------
    end start                          ; Tell MASM where the program ends
```

Directives helps the program will conform to Windows

How to create it? NEXT SLIDE

Form of a MASM program

EX01_Hello.asm

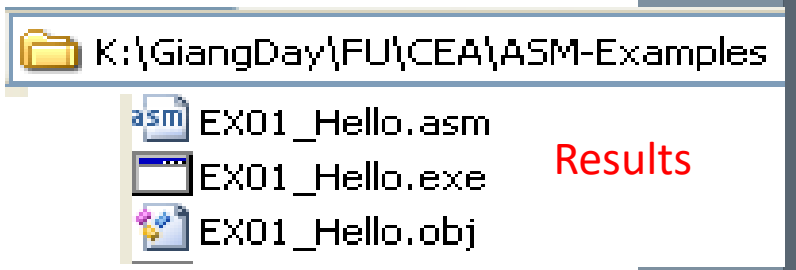
Step 1: Open MASM/ Menu File/ New
Step 2: Copy and paste the code in the next slide to it's editor
Step 3: Save file/EX1_Hello.asm
Step 4: Menu Project/ Console Assemble&Link
Step 5: View results in containing folder
Step 6: Run the program: Click the EX01_Hello.exe
A black window will show then disappear because there is no code to block it.

An Assembly source code is a file whose extension MUST BE .ASM

What is the result of compilation?
You can see them in the folder containing you ASM files

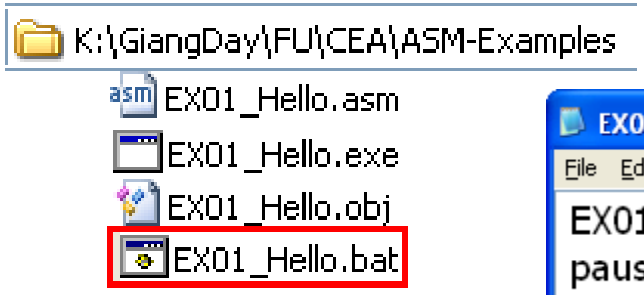
EX01_Hello.asm - Code

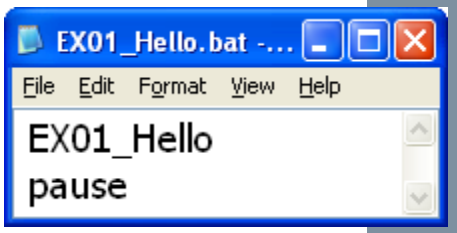
```
; «« Comment begins with ';' to the end of a line
; From masm32\tutorial\console\demo1
;
; Build this with the "Project" menu using
; "Console Assemble and Link"
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

  .486                          ; create 32 bit code
  .model flat, stdcall          ; 32 bit memory model
  option casemap :none          ; case sensitive

  include \masm32\include\windows.inc     ; always first
  include \masm32\macros\macros.asm       ; MASM support macros

; -----------------------------------------------------------------
; include files that have MASM format prototypes for function calls
; -----------------------------------------------------------------
  include \masm32\include\masm32.inc
  include \masm32\include\gdi32.inc
  include \masm32\include\user32.inc
  include \masm32\include\kernel32.inc
; ------------------------------------------------
; Library files that have definitions for function exports
; and tested reliable prebuilt code.
; ------------------------------------------------
  includelib \masm32\lib\masm32.lib
  includelib \masm32\lib\gdi32.lib
  includelib \masm32\lib\user32.lib
  includelib \masm32\lib\kernel32.lib

  .code                 ; Tell MASM where the code starts

  start:                ; The CODE entry point to the program
    print chr$("Hello world!",13,10) ; 13: carriage return, 10: new line
  exit                  ; exit the program

; ------------------------------
  end start             ; Tell MASM where the program ends
```

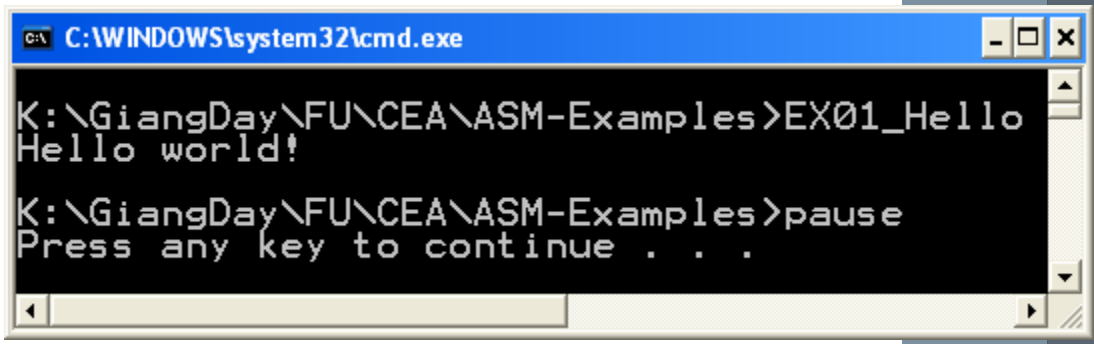


Results

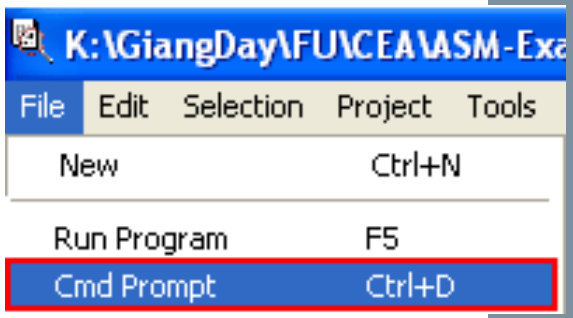
How to run the program and we can see it? ➔
Create EX01_Hello.bat file the run it.







Run a program using the Menu File/Cmd Prompt




```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

K:\GiangDay\FU\CEA\ASM-Examples>dir *.exe
 Volume in drive K is TaiLieu30
 Volume Serial Number is 54D8-4B2E

 Directory of K:\GiangDay\FU\CEA\ASM-Examples

04/16/2015  10:46 AM             2,560 EX01_Hello.exe
04/16/2015  11:41 AM             2,560 EX02_ProcDemo.exe
04/16/2015  01:06 PM             2,560 EX03_Data.exe
04/16/2015  01:46 PM             2,560 EX04_Locals.exe
04/16/2015  03:21 PM             3,072 EX05_Numbers.exe
04/16/2015  04:39 PM             3,072 EX06_Sum.exe
04/16/2015  08:17 PM             3,072 EX07-Swap2.exe
04/16/2015  06:16 PM             3,072 EX07_swap1.exe
04/17/2015  07:17 AM             2,560 MuaXuanChin.exe
               9 File(s)         25,088 bytes
               0 Dir(s)   7,383,755,776 bytes free

K:\GiangDay\FU\CEA\ASM-Examples>EX01_hello
Hello world!

K:\GiangDay\FU\CEA\ASM-Examples>_
```

The command **dir *.exe** will show all exe files stored in the current folder.

Run an application by it's file name (.exe can be ignored)

EX02_ProcDemo.asm

Procedures are a fundamental building block of programs that are build directly into the processor using CALL and RET instructions. This shows how simple it is to do in MASM.

This code is in the directory **masm32\tutorial\console\demo2\Proc.asm**

Procedure syntax

```
    .code                           ; Tell MASM where the code starts

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

start:                              ; The CODE entry point to the program
    call main                       ; branch to the "main" procedure
    exit
; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

main proc
    print chr$("Hi, I am in the 'main' procedure",13,10)
    ret                             ; return to the next instruction after "call"
main endp

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

end start                           ; Tell MASM where the program ends
```

EX02_ProcDemo.asm- Source/Run

```
; From masm32\tutorial\console\demo2
;    Build this with the "Project" menu using
;       "Console Assemble and Link"
; ««««««««««««««««««««««««««««««««««««
   .486                        ; create 32 bit code
   .model flat, stdcall              ; 32 bit memory model
   option casemap :none              ; case sensitive

   include \masm32\include\windows.inc     ; always first
   include \masm32\macros\macros.asm
  ; -----------------------------------------------------------
  ; include files for function calls
  ; -----------------------------------------------------------
   include \masm32\include\masm32.inc
   include \masm32\include\gdi32.inc
   include \masm32\include\user32.inc
   include \masm32\include\kernel32.inc
  ; -----------------------------------------------
  ; Library files that have definitions for function
  ; exports and tested reliable prebuilt code.
  ; -----------------------------------------------
   includelib \masm32\lib\masm32.lib
   includelib \masm32\lib\gdi32.lib
   includelib \masm32\lib\user32.lib
   includelib \masm32\lib\kernel32.lib

    .code                  ; Tell MASM where the code starts

; ««««««««««««««««««««««««««««««

start:                     ; The CODE entry point to the program
   call main               ; branch to the "main" procedure
   exit
; ««««««««««««««««««««««««««««««

main proc
   print chr$("Hi, I am in the 'main' procedure",13,10)
   ret                     ; return to the next instruction after "call"
main endp

; ««««««««««««««««««««««««««««

end start                  ; Tell MASM where the program ends
```

```
K:\GiangDay\FU\CEA\ASM-Examples>EX02_ProcDemo
Hi, I am in the 'main' procedure

K:\GiangDay\FU\CEA\ASM-Examples>_
```

Comments in MASM

Comments are ignored by the assembler

```
; From K:\masm32\tutorial\console\demo3
;                Build this with the "Project" menu using
;                      "Console Assemble and Link"
```

; Comment line

```
comment * «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

A normal component of almost all programs is data. It can be either numeric
data or text data. In MASM you have the distinction between data that is
initialised with a value or UNinitialised data that just reserves the
space to write data to.

Initialised data has this form.

.data
  var1  dd  0                          ; 32 bit value initialised to zero
  var2  dd  125                        ; 32 bit value initialised to 125
  txt1  db  "This is text in MASM",0   ; A zero terminated sequence of TEXT

  array dd 1 2 3 4 5 6 7 8             ; 8 x 32 bit values in sequence
««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««« *
```

COMMENT **delimiter**
[Comment block, extending to the **closing delimiter**]
delimiter

```
.486                                   ; create 32 bit code
.model flat, stdcall                   ; 32 bit memory model
option casemap :none                   ; case sensitive

include \masm32\include\windows.inc    ; always first
include \masm32\macros\macros.asm      ; MASM support macros
```

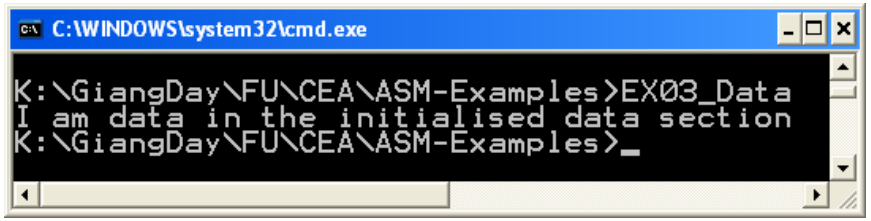
Code

EX03_Data.asm

Print a string declared in the program using the operator OFFSET.
The OFFSET operator tells MASM that the text data is at an OFFSET within the file which means in this instance that it is in the **.DATA** section.

```
    .data
      txtmsg db "I am data in the initialised data section",0

    .code                           ; Tell MASM where the code starts
; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
start:                              ; The CODE entry point to the program
    call main                       ; branch to the "main" procedure
    exit
; »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»

main proc
    print OFFSET txtmsg             ; the "OFFSET" operator tells MASM that the text
                                    ; data is at an OFFSET within the file which means
                                    ; in this instance that it is in the .DATA section

    ret                             ; return to the next instruction after "call"

main endp

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
; »»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»»

end start                           ; Tell MASM where the program ends
```

Data are declared in the **.data** are called as global data



Basic Data Types in MASM32

Type	Abbr	Size (bytes)	Integer range	Types Allowed
BYTE	**DB**	1	-128.. 127	Character, string
WORD	**DW**	2	-32768..32767	16-bit near ptr, 2 characters, double-byte character
DWORD	**DD**	4	-2Gig..(4Gig-1)	16-bit far per, 32-bit near ptr, 32-bit long word
FWORD	**DF**	6	--	32-bit far ptr
QWORD	**DQ**	8	--	64-bit long word
TBYTE	**DT**	10	--	BCD, 10-byte binary number
REAL4	**DD**	4	--	Single-precision floating number
REAL8	**DQ**	8	--	Double-precision floating number
REAL10	**DT**	10	--	10-byte floating point

D: Defined

Initialized data has this form:

```
.data
  var1  dd  0        ; 32 bit value initialized to zero
  var2  dd  125      ; 32 bit value initialized to 125
  txt1  db  "This is text in MASM",0  ; Initialize a  NULL string
  array dd 1,2,3,4,5,6,7,8   ; array of 8 initialized elements
```

Uninitialized data has this form:

```
.data?
  udat1 dd ?      ; Uninitialized single 32 bit space
  buffa db 128 dup (?)          ; buffer 128 bytes
```

EX03_Data.asm - Source

```
; From K:\masm32\tutorial\console\demo3
;               Build this with the "Project" menu using
;                    "Console Assemble and Link"

   .486                          ; create 32 bit code
   .model flat, stdcall                ; 32 bit memory model
   option casemap :none                ; case sensitive

   include \masm32\include\windows.inc    ; always first
   include \masm32\macros\macros.asm      ; MASM support macros
   include \masm32\include\masm32.inc
   include \masm32\include\gdi32.inc
   include \masm32\include\user32.inc
   include \masm32\include\kernel32.inc
    includelib \masm32\lib\masm32.lib
   includelib \masm32\lib\gdi32.lib
   includelib \masm32\lib\user32.lib
   includelib \masm32\lib\kernel32.lib

   .data
     txtmsg db "I am data in the initialised data section",0

   .code                  ; Tell MASM where the code starts
; ««««««««««««««««««««««««««««««««««««««««««««««««««
start:                    ; The CODE entry point to the program
   call main              ; branch to the "main" procedure
   exit
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
main proc
   print OFFSET txtmsg
   ret                  ; return to the next instruction after "call“
main endp
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

end start                 ; Tell MASM where the program ends
```

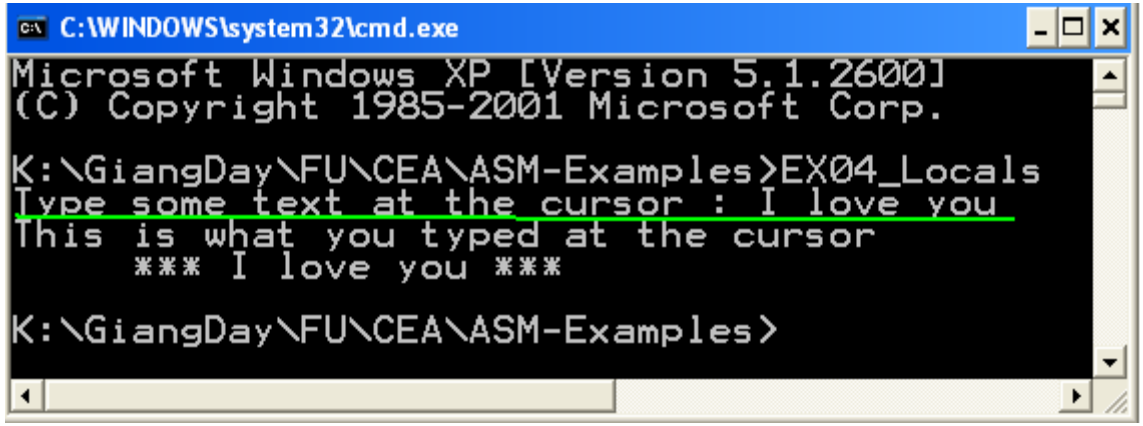
EX04_Locals.asm

How to use of LOCAL variables declared in a procedure?
When the procedure is called, these variables are allocated in program's stack

DECLARE LOCAL VARIABLES

```
LOCAL MyVar:DWORD       ; allocate a 32 bit space on the stack
LOCAL Buffer[128]:BYTE  ; allocate 128 BYTEs of space for TEXT data.
```

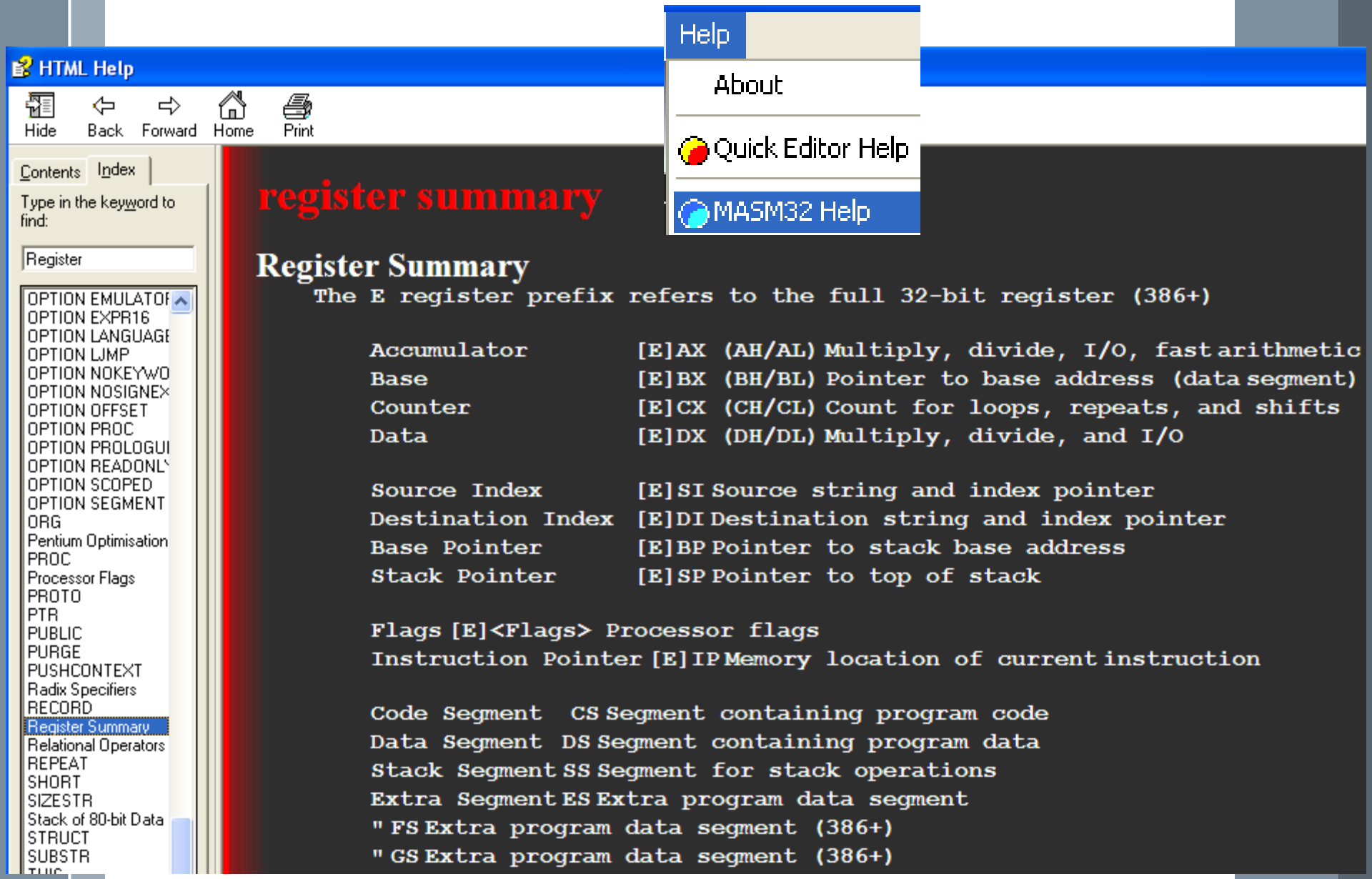
How to PROTOTYPE and implement a procedure along with it's parameters?
How to call user-defined procedure?
How can program receive input from user? → Build-in function: input("warning:")



EX04_Locals.asm (Source code)

```
; Source code From masm32\tutorial\console\demo4\locals.asm
   .486                         ; create 32 bit code
   .model flat, stdcall         ; 32 bit memory model
   option casemap :none         ; case sensitive

   include \masm32\include\windows.inc     ; always first
                                           ; MASM support macros
   show_text PROTO :DWORD       ;  prototype a method + type of parameter

   .code                        ; Tell MASM where the code starts

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

start:                          ; The CODE entry point to the program
    call main                   ; branch to the "main" procedure
    exit

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
                                e a method + type of parameter
main proc
    LOCAL txtinput:DWORD        ; a "handle" for the text returned by "input"
                                code starts
    mov txtinput, input("Type some text at the cursor : ") ; get input string
    invoke show_text, txtinput  ; show inputted string
                                o the program
    ret                          procedure
main endp

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
                                «««««««««««««««««««««««««««««««««««««««««««««««««

show_text proc string:DWORD     or the text returned by "input"
    print chr$("This is what you typed at the cursor",13,10,"     *** ")
                                e cursor : ") ; get input string
    print string                ; show the string at the console
                                ed string
    print chr$(" ***",13,10)
    ret
show_text endp
                                «««««««««««««««««««««««««««««««««««««««««««««««««

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
                                cursor",13,10,"     *** ")
end start                       e console
                                print chr$(      ,13,10)
                                ret
                            show_text endp
                            ; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
                            end start                   ; Tell MASM where the program ends
```

Intel CPU 32-bit Registers



Intel CPU Registers

CS Code Segment
DS: Data Segment
SS: Stack Segment

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

https://msdn.microsoft.com/en-us/library/windows/hardware/ff561499(v=vs.85).aspx

EX05-Numbers.asm

(1) How to receive numbers from user?

Raw data from keyboard are string. The function **sval(string)** will convert num-string to signed number.

(2) How to perform a simple addition using registers

add reg1, reg2 will accumulate value in reg2 to reg1

(2) How to print value in a register/variable to screen

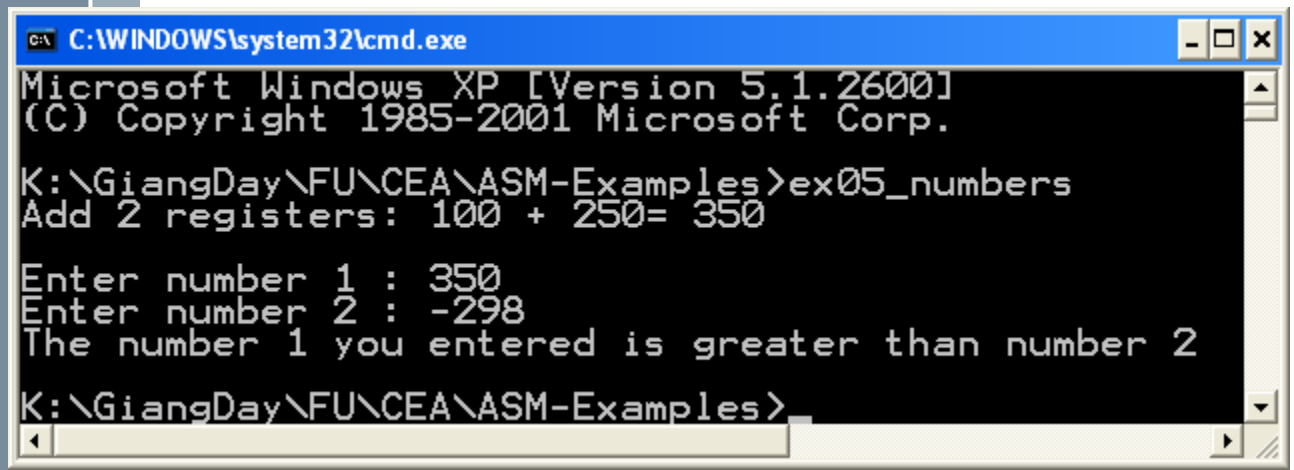
Function str$(number) → num-string

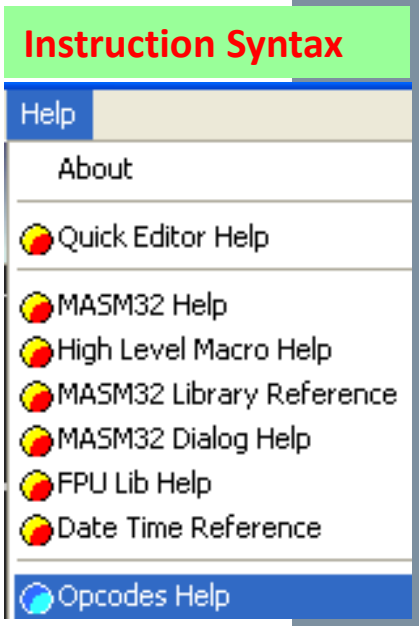
(3) How to compare a memory variable to an immediate number

Use the instruction CMP reg, reg/ CMP reg, var/ CMP var, reg/ CMP mem, immed/ CMP reg, immed (immed= immediate vakue)

(4) How to branching to different labels after camparation?

Use **j**umps: **JE** (equal), JG (greater than), JL (less than)





EX05-Numbrers.asm

Variables Declarations, Input data, Converting data types

```
; EX05_Numbers.asm

; Declare program model and all libraries using only one file

  include \masm32\include\masm32rt.inc

.code
start:                          ; The CODE entry point to the program
    call main                   ; branch to the "main" procedure
    exit
; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

main proc
    LOCAL var1:DWORD            ; 2 DWORD integral variables
    LOCAL var2:DWORD            ;
    LOCAL str1:DWORD            ; a string handle for the input data

  ; test the MOV and ADD instructions
    print chr$("Add 2 registers: 100 + 250= ")
    mov eax, 100                ; copy the IMMEDIATE number 100 into the EAX register
    mov ecx, 250                ; copy the IMMEDIATE number 250 into the ECX register
    add ecx, eax                ; ADD EAX to ECX
    print str$(ecx)             ; show the result at the console
    print chr$(13,10,13,10)     ; 2 empty lines

  ; Input 2 integers
    mov var1, sval(input("Enter number 1 : "))
    mov var2, sval(input("Enter number 2 : "))
```

EX05-Numbers.asm

Comparing and Branching

```
    ; compare 2 variables and process the result
      mov eax, var1                  ; copy var1 to eax
      cmp eax, var2                  ; CMP REG, VAR
      je equal                       ; jump if var1 is equal to 100 to "equal"
      jg bigger                      ; jump if var1 is greater than 100 to "bigger"
      jl smaller                     ; jump if var1 is less than 100 to "smaller"

    equal:
      print chr$("2 numbers you entered are equal.",13,10)
      jmp over

    bigger:
      print chr$("The number 1 you entered is greater than number 2",13,10)
      jmp over

    smaller:
      print chr$("The number 1 you entered is smaller than number 2",13,10)

    over:
      ret
  main endp
  ; <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

  end start                          ; Tell MASM where the program ends
```

EX05-Numbers.asm
Source code

```
; EX05_Numbers.asm
; Declare program model and all libraries using only one file

 include \masm32\include\masm32rt.inc

.code
start:                  ; The CODE entry point to the program
   call main              ; branch to the "main" procedure
   exit
; ««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
main proc
   LOCAL var1:DWORD       ; 2 DWORD integral variables
   LOCAL var2:DWORD       ;
   LOCAL str1:DWORD       ; a string handle for the input data

 ; test the MOV and ADD instructions
   print chr$("Add 2 registers: 100 + 250= ")
   mov eax, 100           ; copy the IMMEDIATE number 100 into the EAX register
   mov ecx, 250           ; copy the IMMEDIATE number 250 into the ECX register
   add ecx, eax           ; ADD EAX to ECX
   print str$(ecx)        ; show the result at the console
   print chr$(13,10,13,10)    ; 2 empty lines

 ; Input 2 integers
   mov var1, sval(input("Enter number 1 : "))
   mov var2, sval(input("Enter number 2 : "))
```

```
 ; compare 2 variables and process the result
   mov eax, var1          ; copy var1 to eax
   cmp eax, var2          ; CMP REG, VAR
   je equal               ; jump if var1 is equal to 100 to "equal"
   jg bigger              ; jump if var1 is greater than 100 to "bigger"
   jl smaller             ; jump if var1 is less than 100 to "smaller"

  equal:
   print chr$("2 numbers you entered are equal.",13,10)
   jmp over

  bigger:
   print chr$("The number 1 you entered is greater than number 2",13,10)
   jmp over

  smaller:
   print chr$("The number 1 you entered is smaller than number 2",13,10)

  over:
   ret
main endp
; ««««««««««««««««««««««««««««««««««««««««««««««««««

end start               ; Tell MASM where the program ends
```

Exercises

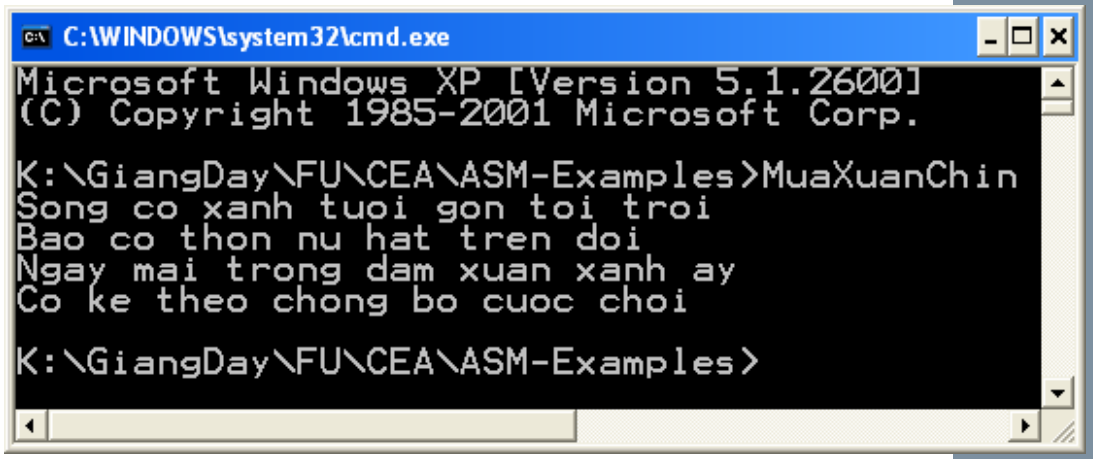
Part 1: Write answers to your notebook

Use the **Opcodes help** of the menu Help, describe syntaxes of following MASM instructions

(1) ADD
(2) SUB
(3) MUL, IMUL
(4) DIV, IDIV. What register will store the remainder ?

Part 2:

Write a MASM program that will print the following cantor of Hàn Mặc Tử



πSummary

- Form of a MASM program
- Variable declarations: DB, DD, DW, …
- Basic input, output operations: print, chr$(…), str$(…)
- Data type conversion: sval(..),
- Procedure with parameters: CALL, INVOKE
- Instructions: MOV, ADD, CMP, JE, JG, JL

Chapter 6
Assembly (2)

Objectives

After studying this chapter, you should be able to:

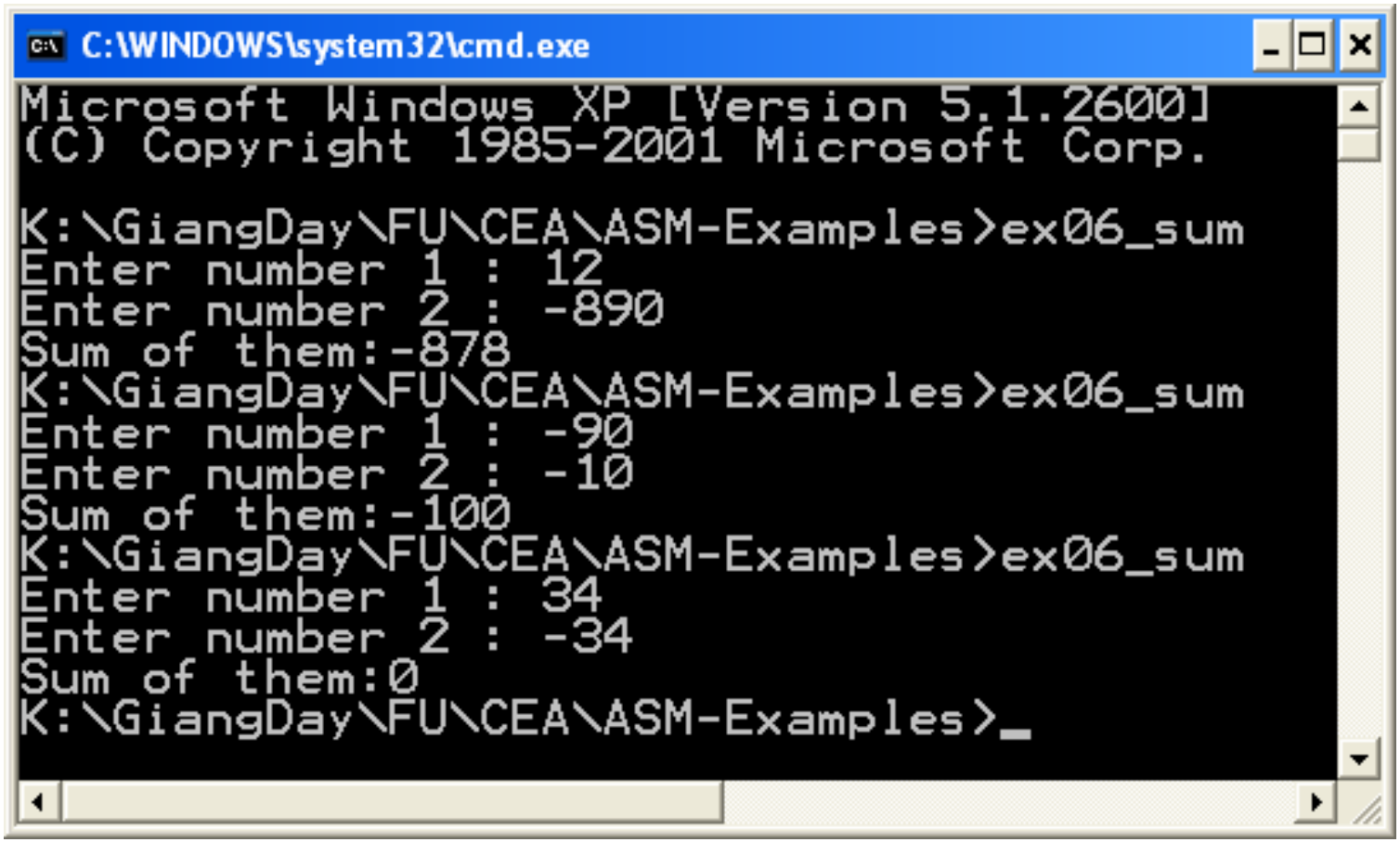
- Perform arithmetic operations
- Access variable's address
- Draw the memory map of a program
- Understand the way procedures work.
- Use a loop operation

Contents

1- Arithmetic Operations

EX06_Adding.asm

Write a MASM program that will accept 2 integers, then sum of them will be print out.



EX06_Adding.asm - Source code

```
; EX06_Sum.asm   Accept 2 integers, sum of them will be printed out
include \masm32\include\masm32rt.inc
sum PROTO :DWORD, :DWORD      ;  prototype a method + 2 parameters
.code
start:                    ; The CODE entry point to the program
   call main               ; branch to the "main" procedure
   exit
; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««
main proc
   LOCAL var1:DWORD          ; 2 DWORD integral variables
   LOCAL var2:DWORD          ;
   LOCAL result:DWORD        ; Result of operation
 ; Input 2 integers
   mov var1, sval(input("Enter number 1 : "))
   mov var2, sval(input("Enter number 2 : "))
 ; Invoke the procedure SUM to compute their sum, result in EAX
   push eax                  ; store EAX to STACK
   invoke sum, var1 , var2
   mov result, eax           ; result = EAX
   pop eax                   ; restore EAX from STACK
```

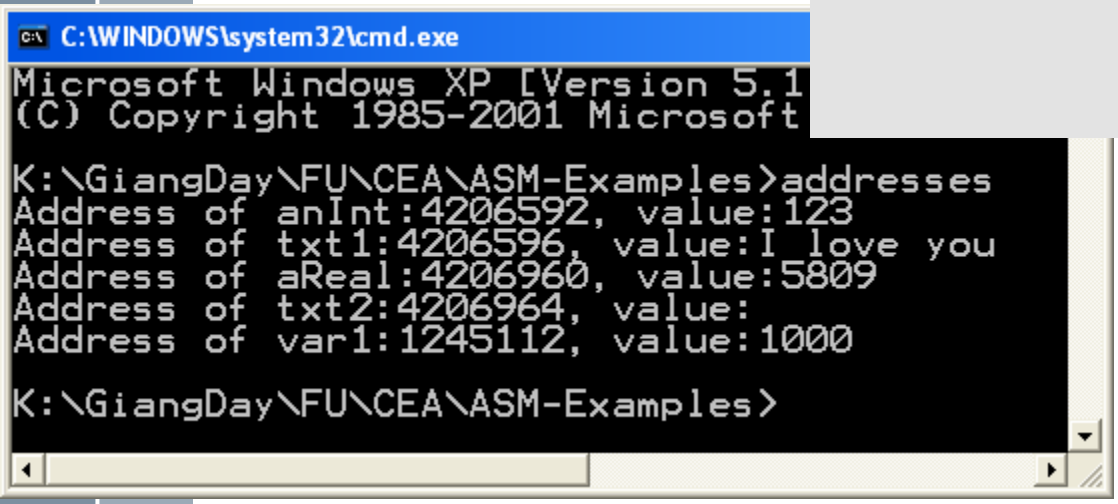
EX06_Adding.asm - Source code

```
 ; Print the result
    print chr$("Sum of them:")
    print str$(result)

    ret
main endp
; «««««««««««««««««««««««««
sum proc v1: DWORD, v2:DWORD
   mov eax, v1          ; eax= v1
   add eax, v2          ; eax = eax + v2 -> Result in eax
   ret
sum endp

end start
```

1- Access Variable's Address

- The following program depicts how to get addresses of variable:
  - The operator OFFSET for global variables
  - The instruction LEA for local variables
  - Draw memory map of a program

Addresses.asm- Access Variable's Address



Data segment (global variables)

(txt2)4206964

(aReal)4206960

(txt1)4206596

(anInt)4206592

(var1)1245112

Stack segment (local variables)

5809

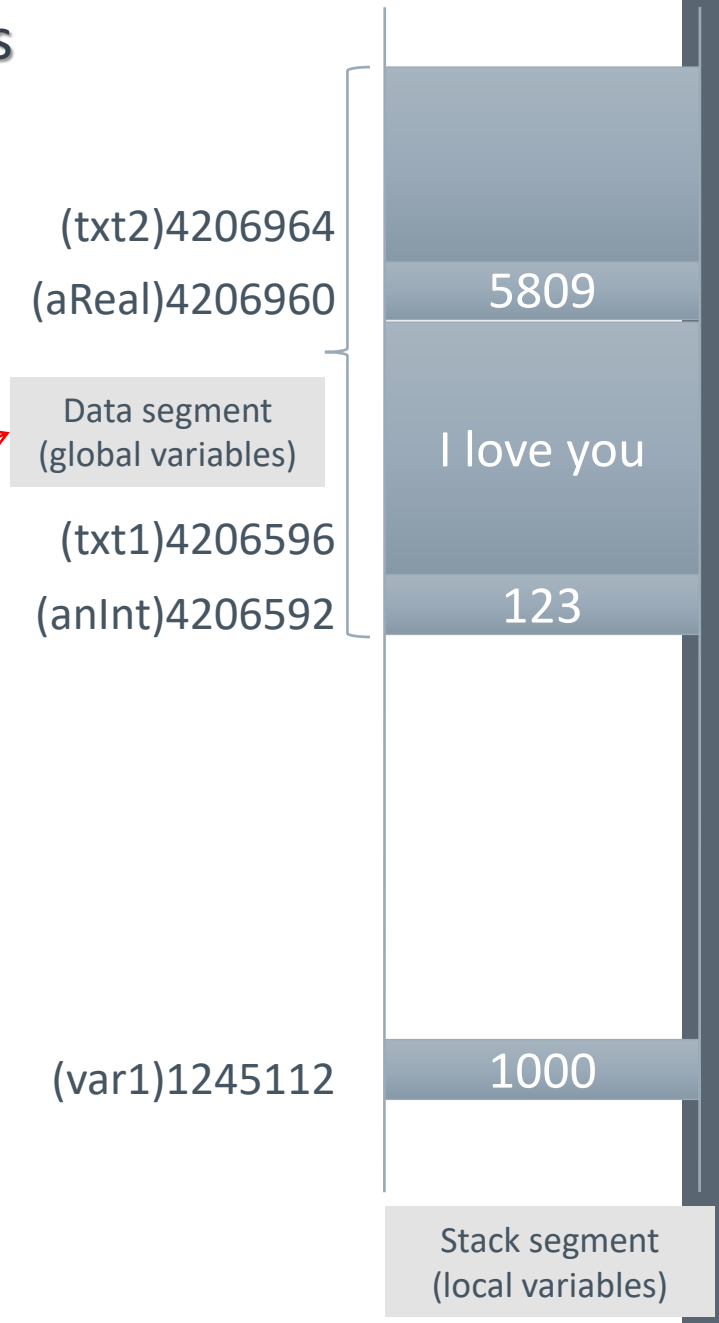
I love you

123

1000

Addresses.asm- Access Variable's Address

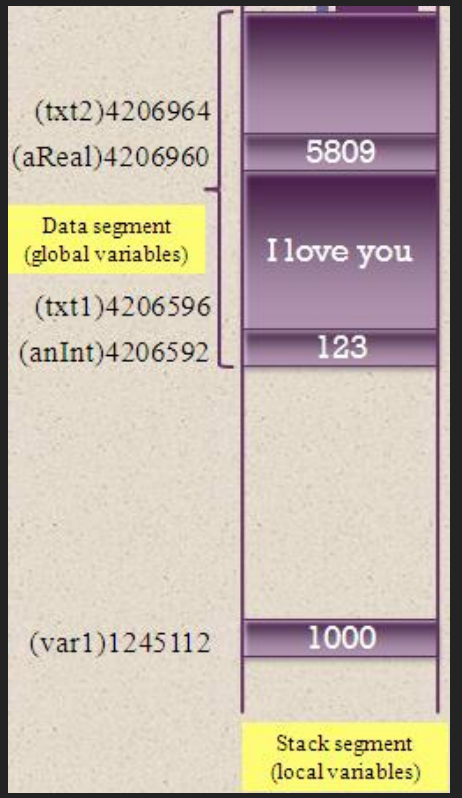
```
; Addresses.asm
; Draw memory of a program

include \masm32\include\masm32rt.inc

.data                ; initialized data
  anInt DD 123
  txt1 db "I love you", 0

.data?               ; Un-initialized data
  aReal DD ?
  txt2  db 128 dup (?)

.code
start:
    call main
    exit
; «««««««««««««««««««««««««««««««««««««««««««««««««««

main proc
    LOCAL var1: DWORD
```



Addresses.asm- Access Variable's Address

```
; Addresses.asm
; Draw memory of a program

include \masm32\include\masm32rt.inc

.data           ; Initialized data
  anInt DD 123
  txt1 db "I love you", 0

.data?          ; Un-initialized data
  aReal DD ?
  txt2  db 128 dup (?)

.code
start:
    call main
    exit
; <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

main proc
    LOCAL var1: DWORD
```

```
    ; Access address and value of anInt
    print chr$("Address of anInt:")
    mov eax, OFFSET anInt ; Operator OFFSET will get address of a global var
    print str$(eax)
    print chr$(", value:")
    print str$(anInt)
    print chr$(13,10)
    ; Access address and value of txt1
    print chr$("Address of txt1:")
    mov eax, OFFSET txt1
    print str$(eax)
    print chr$(", value:")
    print OFFSET txt1
    print chr$(13,10)
    ; Access address and value of aReal
    print chr$("Address of aReal:")
    mov eax, OFFSET aReal
    print str$(eax)
    print chr$(", value:")
    mov aReal, 5809
    print str$(aReal)
    print chr$(13,10)
    ; Access address and value of txt2
    print chr$("Address of txt2:")
    mov eax, OFFSET txt2
    print str$(eax)
    print chr$(", value:")
    print OFFSET txt2
    print chr$(13,10)
    ; Access address and value of local var var1
    mov var1, 1000
    print chr$("Address of var1:")
    lea eax, var1       ; LEA get local variable address
    print str$(eax)
    print chr$(", value:")
    print str$(var1)
    print chr$(13,10)
    ret
main endp
; <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
end start
```

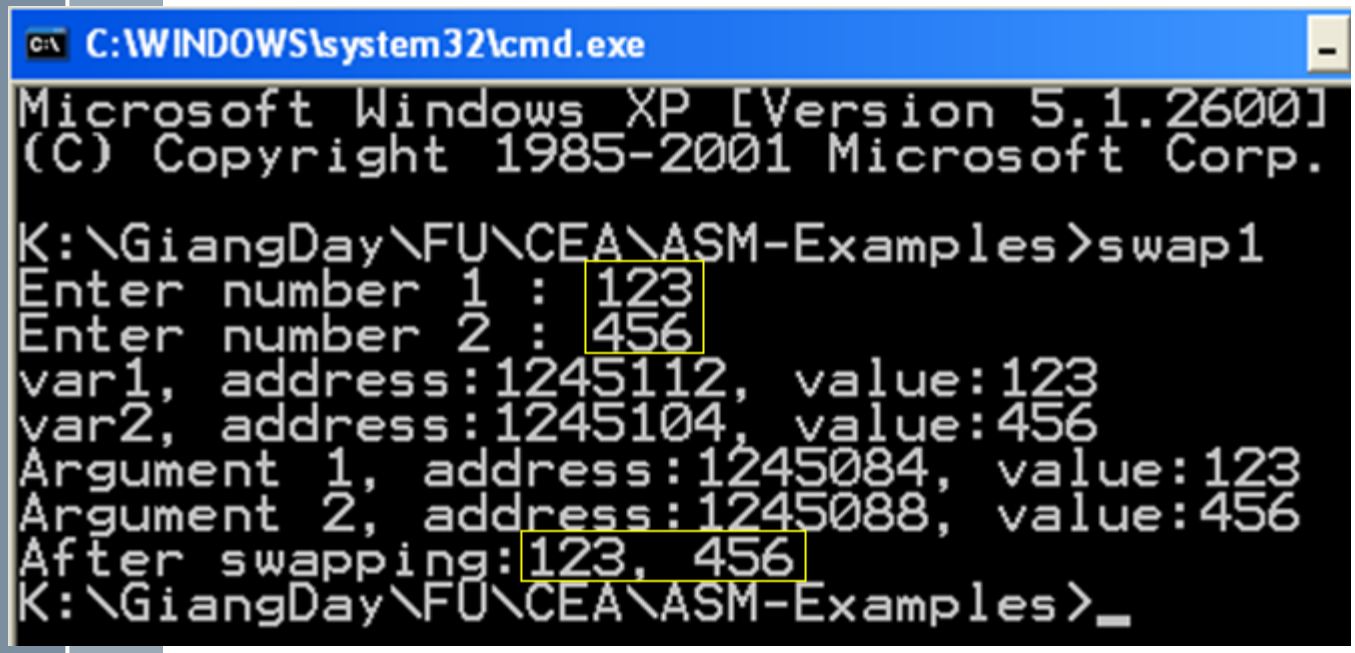


Addresses.asm- Source code

```
; Addresses.asm
; Draw memory of a program

include \masm32\include\masm32rt.inc
 .data          ; initialized data
  anInt DD 123
  txt1 db "I love you", 0
 .data?         ; Un-initialized data
  aReal DD ?
  txt2  db 128 dup (?)
 .code
start:                    ; The CODE entry point to the program
   call main              ; branch to the "main" procedure
   exit
; «««««
main proc
   LOCAL var1: DWORD

 ; Access address and value of anInt
   print chr$("Address of anInt:")
   mov eax, OFFSET anInt ; Operator OFFSET will get address of a
global var
   print str$(eax)
   print chr$(", value:")
   print str$(anInt)
   print chr$(13,10)
 ; Access address and value of txt1
   print chr$("Address of txt1:")
   mov eax, OFFSET txt1
   print str$(eax)
   print chr$(", value:")
   print OFFSET txt1
   print chr$(13,10)
 ; Access address and value of aReal
   print chr$("Address of aReal:")
   mov eax, OFFSET aReal
   print str$(eax)
   print chr$(", value:")
   mov aReal, 5809
   print str$(aReal)
   print chr$(13,10)
 ; Access address and value of txt2
   print chr$("Address of txt2:")
   mov eax, OFFSET txt2
   print str$(eax)
   print chr$(", value:")
   print OFFSET txt2
   print chr$(13,10)
 ; Access address and value of local var var1
   mov var1, 1000
   print chr$("Address of var1:")
   lea eax, var1      ; LEA get local variable address
   print str$(eax)
   print chr$(", value:")
   print str$(var1)
   print chr$(13,10)
   ret
main endp
; »»»»»»»»»»»»»»»»»»»»
end start
```

Swap1.asm

- This program depicts passing values to arguments when calling a procedure
- Program that will accept 2 integers, swap them then print out results.
- This version can not perform requirements successfully because the procedure **swap1** accesses variable directly



Comment:
This program can not swap 2 values

Swap1.asm – Source code

```
; Swap1.asm   Accept 2 integers, swap them, print results
include \masm32\include\masm32rt.inc
swap1 PROTO :DWORD, :DWORD ;  prototype the procedure swap1

.code
start:
    call main
    exit
; «««««««««««««««««««««««««««««««
main proc
    LOCAL var1:DWORD  ; integral variable
    LOCAL pVar1: DWORD   ; address of var1
    LOCAL var2:DWORD  ; integral variable
    LOCAL pVar2: DWORD  ; address of var2

  ; Input 2 integers
    mov var1, sval(input("Enter number 1 : "))
    mov var2, sval(input("Enter number 2 : "))

  ; Get addresses of var1, var2 to pVar1, pVar2
    lea eax, var1
    mov pVar1, eax
    lea eax, var2
    mov pVar2, eax

  ; Output information of var1, var2
    print chr$("var1, address:")
    print str$(pVar1)
    print chr$(", value:")
    print str$(var1)
    print chr$(13,10)
    print chr$("var2, address:")
    print str$(pVar2)
    print chr$(", value:")
    print str$(var2)
    print chr$(13,10)

    ; Invoke the procedure SWAP1 to swap 2 values inputted
      push eax          ; store EAX to STACK
      push ebx          ; store EBX to STACK
      invoke swap1, var1 , var2
      pop ebx           ; restore EBX, EAX from STACK
      pop eax
    ; Print the result
      print chr$("After swapping:")
      print str$(var1)
      print chr$(", ")
      print str$(var2)

      ret
main endp
; «««««««««««««««««««««««««««««««««««««««««««««««««««««
swap1 proc v1: DWORD, v2:DWORD
    ; Print out information of arguments
      print chr$("Argument 1, address:")
      lea eax, v1
      print str$(eax)
      print chr$(", value:")
      print str$(v1)
      print chr$(13,10)
      print chr$("Argument 2, address:")
      lea eax, v2
      print str$(eax)
      print chr$(", value:")
      print str$(v2)
      print chr$(13,10)
    ; swap values
      mov eax, v1 ; eax= v1
      mov ebx, v2 ; eax= v1
      mov v1, ebx
      mov v2, eax
      ret
swap1 endp

end start
```

Swap1.asm – Memory Map when the procedure Swap1 is called

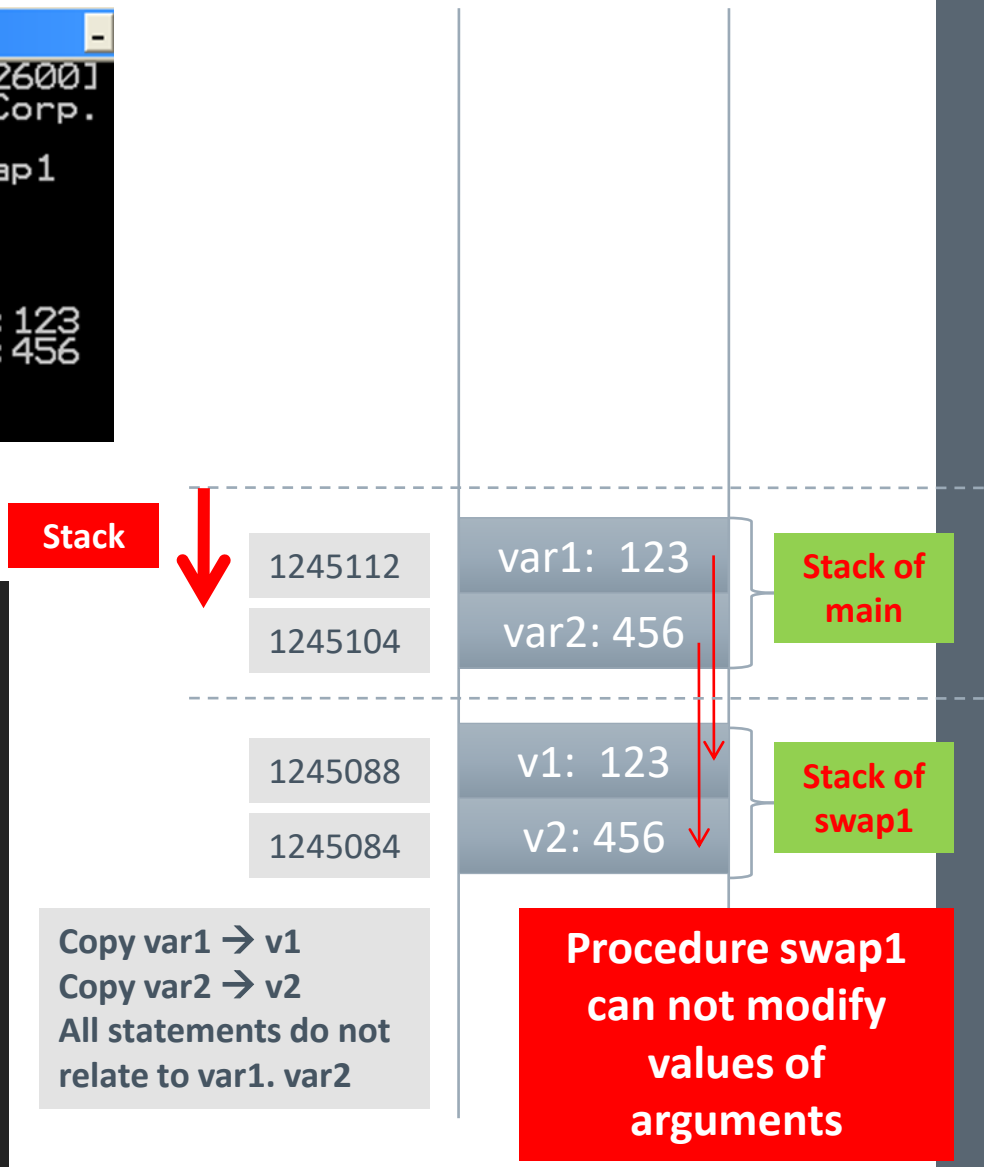
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

K:\GiangDay\FU\CEA\ASM-Examples>swap1
Enter number 1 : 123
Enter number 2 : 456
var1, address:1245112, value:123
var2, address:1245104, value:456
Argument 1, address:1245084, value:123
Argument 2, address:1245088, value:456
After swapping:123, 456
K:\GiangDay\FU\CEA\ASM-Examples>_
```

From **main**

```
invoke swap1, var1 , var2
```

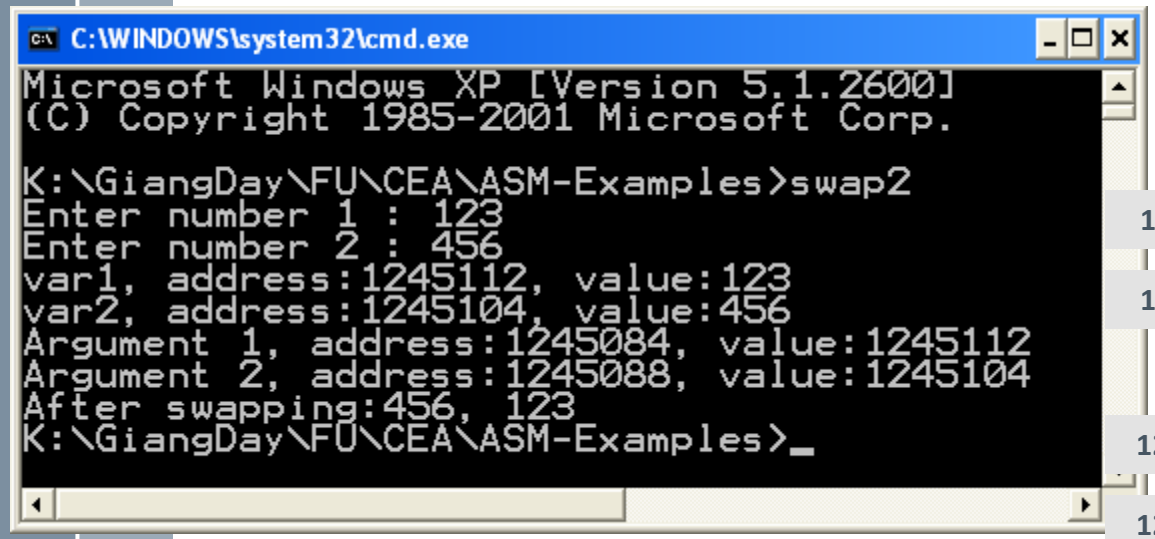
call

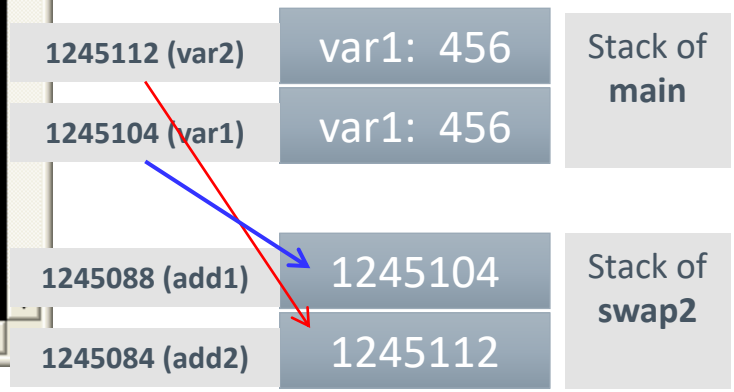
```
swap1 proc v1: DWORD, v2:DWORD
   ; Print out information of arguments
     print chr$("Argument 1, address:")
     lea eax, v1
     print str$(eax)
     print chr$(", value:")
     print str$(v1)
     print chr$(13,10)
     print chr$("Argument 2, address:")
     lea eax, v2
     print str$(eax)
     print chr$(", value:")
     print str$(v2)
     print chr$(13,10)
   ; swap values
     mov eax, v1 ; eax= v1
     mov ebx, v2 ; eax= v1
     mov v1, ebx
     mov v2, eax
     ret
swap1 endp
```



Swap2.asm

- This program will repair the failure of the previous program in which a procedure is declared using pointers (address of data)





To change values of var1 and var2

t1= value at add1
t2= value at add2
Value at add1 = t2
Value at add2= t1

Swap2.asm – Source code

```
; Swap2.asm   Accept 2 integers, swap them, print results
include \masm32\include\masm32rt.inc

swap2 PROTO : PTR DWORD, : PTR DWORD ;  parameters are addresses

.code
start:
    call main
    exit
; «««««««««««««««««««««««««««««««««
main proc
    LOCAL var1:DWORD  ; integral variable
    LOCAL pVar1: DWORD   ; address of var1
    LOCAL var2:DWORD  ; integral variable
    LOCAL pVar2: DWORD  ; address of var2

  ; Input 2 integers
    mov var1, sval(input("Enter number 1 : "))
    mov var2, sval(input("Enter number 2 : "))

  ; Get addresses of var1, var2 to pVar1, pVar2
    lea eax, var1  ; var1, var2 are local --> use LEA
    mov pVar1, eax
    lea eax, var2
    mov pVar2, eax
```

Swap2.asm – Source code

```
; Output information of var1, var2
  print chr$("var1, address:")
  print str$(pVar1)
  print chr$(", value:")
  print str$(var1)
  print chr$(13,10)
  print chr$("var2, address:")
  print str$(pVar2)
  print chr$(", value:")
  print str$(var2)
  print chr$(13,10)

; Invoke the procedure SWAP1 to swap 2 values inputted
  push eax          ; store EAX to STACK
  push ebx          ; store EBX to STACK
  invoke swap2, pVar1 , pVar2
  pop ebx           ; restore EBX, EAX from STACK
  pop eax

; Print the result
  print chr$("After swapping:")
  print str$(var1)
  print chr$(", ")
  print str$(var2)

  ret
main endp
```

Swap2.asm – Source code

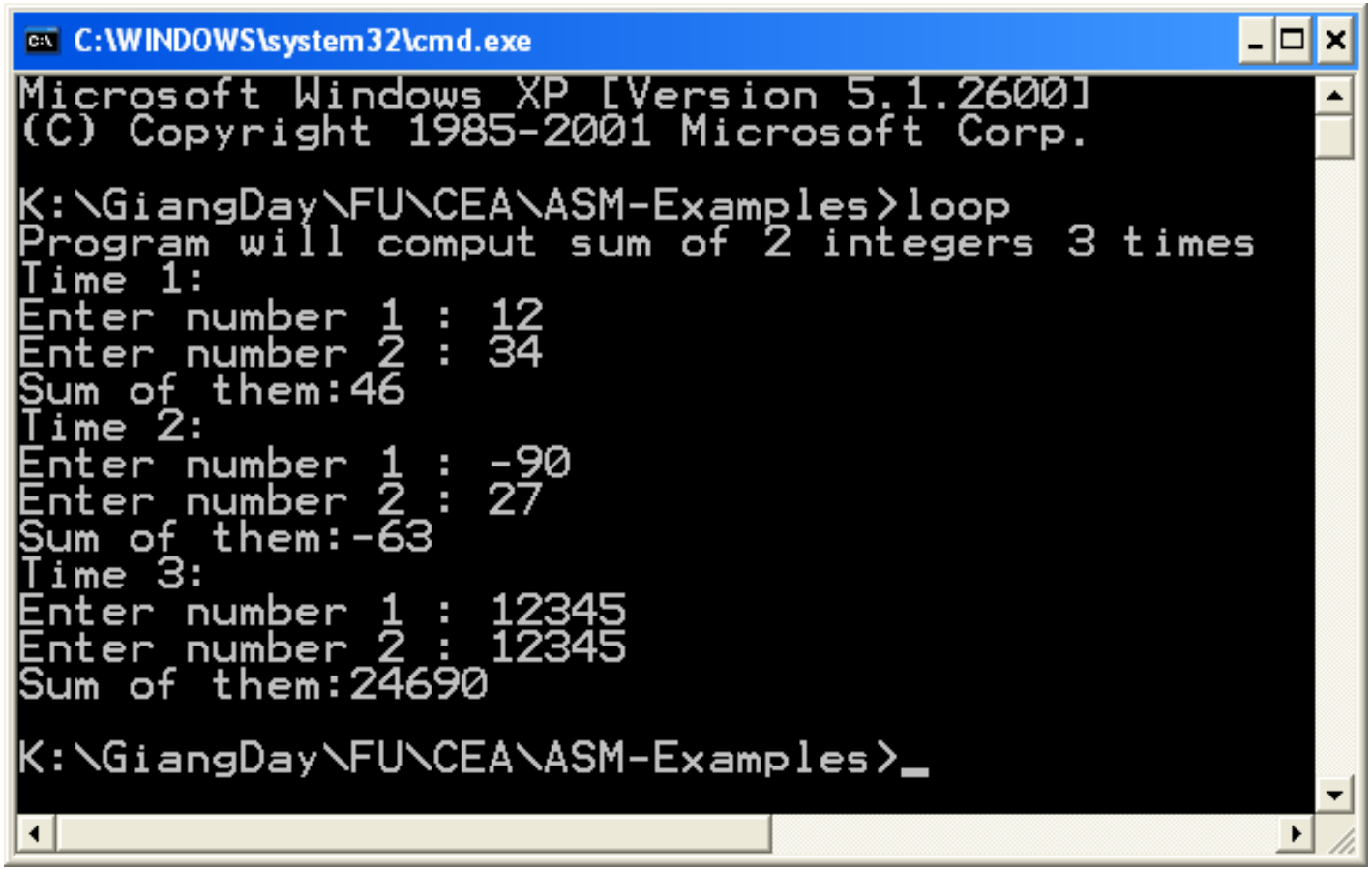
```
   ; Print the result
     print chr$("After swapping:")
     print str$(var1)
     print chr$(", ")
     print str$(var2)

     ret
main endp
; «««««««««««««««««««««««««««««««««««««««««««««««««««««
swap2 proc add1: PTR DWORD, add2: PTR DWORD
   ; Print out information of arguments
     print chr$("Argument 1, address:")
     lea eax, add1
     print str$(eax)
     print chr$(", value:")
     print str$(add1)
     print chr$(13,10)
     print chr$("Argument 2, address:")
     lea eax, add2
     print str$(eax)
     print chr$(", value:")
     print str$(add2)
     print chr$(13,10)
   ; swap values
     mov edx, add1  ; You must access value at address using a regis
     mov eax, [edx] ; eax= value at add1
     mov edx, add2
     mov ebx, [edx] ; ebx= value at add2
     mov edx, add1
     mov [edx], ebx  ; value at add1= ebx
     mov edx, add2
     mov [edx], eax  ; value at add2= eax
     ret
swap2 endp

end start
```

To change values of var1 and var2

t1= value at add1
t2= value at add2
Value at add1 = t2
Value at add2= t1

Loop.asm

The following program depicts a way to use loop.
Program will perform 3 times, for each time, 2 integers will be accepted then sum of them is print out.



Loop.asm – Source code

```
; Loop.asm  Loop 3 times, for each time: Accept 2 integers,
; sum of them will be printed out

 include \masm32\include\masm32rt.inc

; Prototype the sum function, 2 parameters
 sum PROTO :DWORD, :DWORD

.code
start:
    call main
    exit
; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««

main proc
    LOCAL var1:DWORD        ; 2 DWORD integral variables
    LOCAL var2:DWORD
    LOCAL result:DWORD      ; Result of operation
    LOCAL COUNT: DWORD      ; count the loop

  ; initialize the loop
    mov COUNT, 3
    print chr$("Program will comput sum of 2 integers ")
    print str$ (COUNT)
    print chr$(" times", 13,10)
```

Loop.asm – Source code

```
CONTD:                        ; Label for loop
    CMP COUNT, 0              ; While COUNT>3 DO
    je STOP                   ; if COUNT =0, program terminates
    print chr$("Time ")
    mov eax, 4                ; 4-3 =1, 4-2=2, 4-1=3
    SUB eax, COUNT
    print str$(eax)
    print chr$(":", 13, 10)
; Input 2 integers
    mov var1, sval(input("Enter number 1 : "))
    mov var2, sval(input("Enter number 2 : "))
; Invoke the procedure SUM to compute their sum
    invoke sum, var1 , var2
    mov result, eax   ; result = eax
; Print the result
    print chr$("Sum of them:")
    print str$(result)
    print chr$(13,10)
    DEC COUNT                 ; COUNT = COUNT -1
    JMP CONTD                 ; repeat

STOP:
    ret

main endp

; «««««««««««««««««««««««««««««««««««««««««««««««««««««««««««
sum proc v1: DWORD, v2:DWORD
   mov eax, v1 ; eax= v1
   add eax, v2 ; eax = eax + v2 -> Result in eax
   ret
sum endp

end start
```

Exercises

Develop a MASM program that will accept 2 integers then print out their sum, difference, product, quotient and remainder. (Refer to the EX06_sum.asm as a sample)

Develop a MASM program that will accept 3 integers then print out their sum.

Develop a program that will perform n times, for each time, 2 integers will be accepted then sum of them is print out. Value of n is received from user.

πSummary

- Arithmetic operations
- Accessing variable's address
- Drawing the memory map of a program
- The way procedures work.
- Using loops