**Lab 08: Defining and Using Procedures**

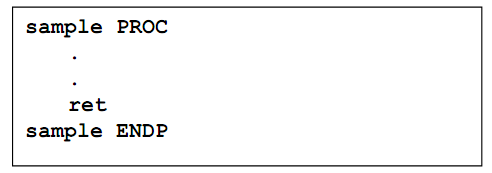
**OBJECTIVE**

To learn how to make procedures and perform procedure calls.

**Procedures** also known as *subroutines* or *functions*is a named block of statements that ends in a ‘return’ statement. ‘Procedures’ allow large problems to be divided into smaller parts that can be called more than once. This will make the program more manageable. A procedure in Assembly is equivalent to a Java or C++ function. CPU uses the runtime **stack** to track the location of procedures. **Main proc** is the main procedure that ends with **main endp**.

**Defining a Procedure using PROC directive:**

A ‘Procedure’ is declared using the **PROC** and **ENDP** directives. A procedure must be assigned a name (a valid identifier). When you create a procedure, end it with a **RET** (return) instruction. **RET** forces the CPU to return to the location from where the procedure was called. Following is the example of a procedure named ‘Sample’.

****

**Labels in Procedures:**

By default, **labels** for JMP and LOOP instructions are visible only within the procedure in which they are declared. For example the **label** named destination must be located in the same procedure as the JMP instruction: **JMP** destination

Because procedures have an automated way of returning and adjusting the runtime stack. So it is not a good idea to jump or loop outside the current procedure. Otherwise the runtime stack can easily become corrupted.

**What do Procedures take and return?**

Each procedure has a list of input parameters e.g. from registers or memory.

Each procedure returns a value stored in a register specified by the procedure.

A procedure may have a list of any special requirements called **pre-conditions**, that must be satisfied before the procedure is called e.g. passing values.

**Example:** Sum of three integers procedure.

Comments: The procedure named **SumOf** calculates the sum of three 16-bit integers. We will assume that relevant integers are assigned to AX, BX and CX before the procedure is called. The procedure returns the sum in AX:

**SumOf** PROC

Add ax, bx

Add ax,cx

Ret

**SumOf** ENDP

**CALL and RET Instructions:**

The **CALL** instruction calls a procedure and does the following:

* It pushes offset of next instruction (return address) on the stack.
* Copies the address of the called procedure into IP (instruction pointer).

The **RET** instruction returns from a procedure and:

* Pops top of the stack (return address) into IP.

**Passing Register Arguments to Procedures:**

A good procedure might be useful in many different programs. But it could only be used only once if it refers to specific variable names.

Passing *arguments (input parameters)*can make a procedure flexible, because parameter values can change at runtime.

Place all ‘parameters’ in a **main** accessible storage area, then call the procedure

Two common methods of ‘parameter passing’ are:

* General Purpose Registers (AX, BX, CX, DX).
* Stack (memory) method.

**Code-01:** Calling the **SumOf** three integers procedure.

.model small

.stack 100h

.data

**theSum** dw ?

.code

Main Proc

Mov ax,1000h ; argument

Mov bx,2000h ; argument

Mov cx,3000h ; argument

**Call SumOf** ; AX = AX + BX + CX

Mov theSum,ax

Main endp

**SumOf** PROC

Add ax, bx

Add ax,cx

Ret

SumOf ENDP

**Code-02:** A procedure to generate a new line.

.model small

.stack 100h

.data

.code

Main proc

Mov ah,2

Mov dl,’A’

Int 21h

**Call NewLine**

Mov ah,2

Mov dl,’B’

Int 21h

Mov ah,4ch ; terminate program

Int 21h

Main endp

**NewLine** proc

Mov ah,2

Mov dl,0dh

Int 21h

Mov dl,0ah

Int 21h

Ret

NewLine endp

**Code-03:** A procedure to display the two-digit product of two integers.

.model small

.stack 100h

.data

.code

Main proc

Mov al,8

Mov bl,2

Mul bl ; AX = AL \* BL = 8 \* 2 = 16

**Call Display**

Mov ah,4ch

Int 21h

Main endp

**Display** proc

Mov cl,10

Div cl

Mov ch,ah ; remainder moved to CH to avoid loss of contents

Mov ah,2

Mov dl,al ; display quotient first

Add dl,30h

Int 21h

Mov ah,2

Mov dl,ch ; display remainder second

Add dl,30h

Int 21h

Ret

Display endp

**Activities**: To do list.

* Write a Procedure which when called, prints time on Console. (Hint given below)

|  |  |  |
| --- | --- | --- |
| AH = **2Ch**; Get system time  AH = **2Ah**; Get system date | Get **System Time**:  Mov ah,2ch  Int 21h  CH = 0-23 hour; CL = 0-59 minutes; DH = 0-59 second DL = 1/100 seconds (0-99) | Get **System Date**:  Mov ah,2ah  Int 21h  DL = date; DH = month;  AL = day of week;  CX = current year |

* Write a Procedure which takes 5 numbers as argument. Calculate sum of numbers and return the sum. Prints the sum in main procedure. You have to use Stack for Passing and Retrieving Arguments.

**< The End >**