Laboratory Exercise #3-3
Sub routines and DOS Interrupts

Target Course Outcome:
CO2: Designs a microprocessor-based firmware integrating a microprocessor with supporting peripherals and devices.

Objectives:
To understand the concept of subroutines and use it in assembly language programs.
To know the different DOS interrupts and use such interrupts in assembly language programs.

Tools Required:
Emu8086 Emulator

Part 1:

Subroutines

A program is made up of instructions to solve a given problem. In each program, it is often needed to perform a particular sub-task many times on different data values. So, we split the program into smaller units which solve a particular part of the problem. These task specific sub-units are termed as functions, procedures or subroutines. In assembly language, we use the word subroutine for all subprograms to distinguish between functions used in other programming languages and those used in assembly languages. The block of instructions that constitute a subroutine can be included at every point in the main program when that task is needed. However, this would result in unnecessary waste of memory space. Rather, only one copy of the instructions that constitute the subroutine is placed in memory and can be accessed repeatedly.

Subroutines are groups of instructions that usually perform one task. These tasks may be reused as often as necessary and may be called from anywhere in the program. Memory is saved by using procedures, but one disadvantage is the longer execution time needed for the computer to link to the procedure and to return to the location where the procedure was called.

Components of Subroutines

An assembly language routine has:

1. An entry point. The location of the first instruction in the routine.
2. Parameters. The list of registers or memory locations that contain the parameters for the routine.
3. Return values. The list of registers or memory locations to save the results.

4. Working storage. The registers or memory location required by the routine to perform its task.

Calling Subroutines

There are two ideas behind a subroutine:
1. You should be able to call the subroutine from anywhere.
2. Once the subroutine is complete, it should return to the place that called the subroutine.

There are special instructions for transferring control to subroutines and restoring control to the main program. The instruction that transfers the control is usually termed as call, jump or branch to subroutine. The calling program is called Caller and the subroutine called is known as Callee. The instruction that transfers control back to the caller is known as Return. Calling a subroutine requires a deviation from the default sequential execution of instructions. When a program branches to a subroutine, the processor begins execution of the instructions that make up the subroutine and branch to the subroutine by modifying the Program Counter (PC).

Two instructions are used with procedures: CALL, used to execute the procedure, and RET, used to return program execution to the point following the CALL statement.

```
ORG 100H

  MOV AH,9H
  MOV DX,OFFSET MESS
  INT 21H
  CALL DELAY

RET

MESS DB 'Hello World!$'

DELAY:
  PUSH CX
  MOV CX,00FFH
  AGAIN:
    LOOP AGAIN
  POP CX
RET
```

The main procedure is identified by the last statement of the program, therefore the order or arrangement of all procedures within the program does not matter. In the sample code below, the main procedure is at the top followed by subroutines.

Interrupts

Interrupts can be seen as several functions. These functions make the programming much easier, instead of writing a code to print a character you can simply call the interrupt and it will do everything foryou. There are also interrupt functions that work with disk drive and other hardware. We call such functions software interrupts. Interrupts are also triggered by different hardware; these are called hardware interrupts. Currently we are interested in software interrupts only.

To make a software interrupt there is an INT instruction, it has very simple syntax:
INT value, where value can be a number between 0 to 255 (or 0 to 0FFh), generally we will use hexadecimal numbers. You may think that there are only 256 functions, but that is not correct. Each interrupt may have sub-functions. To specify a sub-function AH register should be set before calling interrupt. Each interrupt may have up to 256 sub-functions (so we get 256 * 256 = 65536 functions). In general AH register is used, but sometimes other registers maybe in use.

Generally other registers are used to pass parameters and data to sub-function.

DOS INT 21H

INT 21H is provided by DOS, and it is stored in DRAM when the operating system is loaded. The user can invoke this interrupt to perform several useful functions, like inputting data from keyboard, and outputting data to monitor. These interrupt subroutines may be used by issuing a software interrupt call (**INT *type***). The user will have to identify which function is being used by setting the AH register to a specific value. Other registers may also need to be modified. Following are the descriptions of several of the most common INT 21H functions.

INT 21H Function 01H: Inputting a single character from the keyboard with echo.

AH = 01H
AL = inputted ASCII character code

Code example to input a single character from keyboard and to echo it to the display.

```
MOV AH,01H
INT 21H
```

INT 21H Function 02H: Outputting a single character to the monitor.

AH = 02H
DL = ASCII character code to be displayed

INT 21h Function 07H: Character input without echo to AL.
If there is no character in the keyboard buffer, the function waits until any key is pressed.

AH=07
INT 21H

INT 21H Function 09H: Outputting a string terminated with $ to the monitor.

AH = 09H
DX = String address

INT 21H Function 4CH: Terminate a process (EXIT)

AH = 4CH
AL = binary return code

INT 21H Function OAH: Input string of data from the keyboard
AH = 0AH

Input string to **DS:DX**, fist byte is buffer size, second byte is number of chars actually read. This function does **not** add '$' in the end of string. To print using **INT 21h** / **AH=9** you must set dollar character at the end of it and start printing from address **DS:DX + 2**.

Part II
Write a program to implement a subroutine and use any of the INT 21 functions.

Activity #1
Using subroutines and INT 21 function.

Procedure:
1. Start Emu8086 by selecting its icon from the start menu, or by running Emu8086.exe
2. Create a new file and label it as Exp3-3.asm
3. In the editor window, write the following instructions.

```
ORG 100H
   MOV DX, OFFSET BUFFER
   MOV AH, 0AH
   INT 21H
   CALL PRINT
RET

PRINT:

  XOR BX, BX
  MOV BL, BUFFER[1]
  MOV BUFFER[BX+2], '$'
  MOV DX, OFFSET BUFFER + 2
  MOV AH, 9
  INT 21H
RET

BUFFER DB 10, ?, 10 DUP(' ')
```

4. Press the Compile and Emulate button or press F5 key on your keyboard. The emulator window should open with this program loaded, click the Single Step button and watch the register values.
5. Write the value of the registers in Canvas after each instruction execution as seen on the emulator window.

Copyright Information

Author : Rosana J. Ferolin (rjferolin@usc.edu.ph)
Date of release : August 7, 2020
Version : 1.0

Change log:

Date	Version	Author	Changes
Aug. 7, 2020	1.0	Rosana J. Ferolin	Initial Draft
Sep. 3, 2025	2.0	Marlowe Edgar C. Burce	Corrected coding sample errors, revised code and procedures