Assembly Language Lab # 9

Stacks and Subroutines

Eng. Doaa Abu Jabal
Assembly Language Lab # 9

Stacks and Subroutines

Objective:
To learn how to use the stack and write assembly procedures.

Stacks:
The stack is part of memory used for temporary storage of addresses and data. It's LIFO structure, the stack sorts data in 2 bytes. The SS and SP registers point to the top of the stack like this: SS:[SP]. So the SS register is the segment and the SP register contains the offset. The **PUSH** instruction is used to place values on the stack and the **POP** instruction is used to remove values from the stack.

- When we dealing with stack in the program we write:
  `.stack`

- To determine the size of the stack, which we want to use it, we put it as follows:
  `.stack 100h`

1. **PUSH Instruction**
The PUSH instruction first decrements ESP and then copies either a 16- or 32-bit source operand into the stack.
   A 16-bit operand causes ESP to be decremented by 2.
   A 32-bit operand causes ESP to be decremented by 4.

- There are three instruction formats:
  PUSH r /m16
  PUSH r /m32
  PUSH imm32
2. **POP Instruction**

The POP instruction first copies the contents of the stack element pointed to by ESP into a 16- or 32-bit destination operand and then increments ESP. If the operand is 16 bits, ESP is incremented by 2; if the operand is 32 bits, ESP is incremented by 4.

- There are two instruction formats:
  - POP r/m16
  - POP r/m32

3. **PUSHFD and POPFD Instructions**

The PUSHFD instruction pushes the 32-bit EFLAGS register on the stack, and POPFD pops the stack into EFLAGS:

```
pushfd
popfd
```

4. **PUSHAD, PUSHA, POPAD, and POPA**

- The PUSHAD instruction pushes all of the 32-bit general-purpose registers on the stack in the following order: EAX, ECX, EDX, EBX, ESP, EBP, ESI, and EDI.
- The POPAD instruction pops the same registers off the stack in reverse order.
- The PUSHA instruction, pushes the 16-bit general-purpose registers (AX, CX, DX, BX, SP, BP, SI, DI) on the stack in the order listed.
- The POPA instruction pops the same registers in reverse order.

**Note:** we will use the following interrupt services:

- **Service 01h:** DOS get character function
  - `mov ah,01h ; returns ASCII code of character to AL`
  - `int 21h ; and echo it to the monitor`

- **Service 02h:** DOS print character function
  - `mov ah,02h`
  - `mov dl,ASCII# ; ASCII code of character for print in DL`
  - `int 21h`

- **Service 08h:** Get character without echo
  - `mov ah,08h ; returns ASCII code of character to AL`
  - `int 21h ; but don`t echo it to the monitor`
Lab work:

**Example 1:**
Write an assembly language program asks the user to enter a password formed from 8 characters. The program prints the password as stars on the screen. If the password is right, the program should print 'Correct Password'. Else, it will print 'Incorrect Password'.

```
.dosseg
.model small
.data
pwd db 'assembly'
msg1 db 'Correct Password',0ah,0dh,'$'
msg2 db 'Incorrect Password',0ah,0dh,'$'
.code
main: mov ax,0data
mov ds, ax
mov bx, offset pwd
mov ex, 8
x: mov ah, 8
int 21h
push ax
push [bx]
inc bx
mov ah, 2
mov dl, 'x'
int 21h
loop x
mov ex, 8
v: pop ax
```

```
pop bx
cmp al, hl
jne error
loop y
mov ah, 9
mov dx, offset msg1
int 21h
jmp exit
error: mov ah, 9
mov dx, offset msg2
int 21h
exit: mov ah, 4ch
int 21h
End main
```
Subroutines:

A subroutine is a special part of the program that can be called for execution from any point in the program. The subroutine is written to provide a function that must be performed frequently at various points in the main program. Whenever the function must be performed, a single instruction is inserted into the main body of the program to CALL the subroutine. RET instruction must be included at the end of the subroutine to return to the main program.

- **The CALL instruction**: calls a procedure
  - pushes offset of next instruction on the stack
  - copies the address of the called procedure into EIP

- **The RET instruction**: returns from a procedure
  - pops top of stack into EIP

Suggested documentation for each procedure:

- A description of all tasks accomplished by the procedure.
- **Receives**: A list of input parameters; state their usage and requirements.
- **Returns**: A description of values returned by the procedure.
- **Requires**: Optional list of requirements called preconditions that must be satisfied before the procedure is called.

Note:

To convert a hexadecimal digit (x) to its ASCII code (y):

\[
y = \begin{cases} 
  x+30h & \text{if } 0<x<9 \\
  x+37h & \text{if } A<x<F \\
  \text{ERROR} & \text{else}
\end{cases}
\]
Lab work:

Example 2:

This program calls a subroutine that finds the largest number in a vector with N elements:
**Homework:**

**H.W 1:**

Write an assembly program that gets a character from Keyboard and displays its ASCII.

**H.W 2:**

Write a procedure that sets the Zero flag if the 32-bit integer passed in the EAX register is prime. (A prime number is evenly divisible by only itself and 1) Optimize the program's loop to run as efficiently as possible. Your program should prompt the user for a number and then display a message indicating whether or not the number is prime. The program should then ask for another number from the user. Continue the loop in this fashion until the user enters a prearranged value such as -1.